TECHNOLOGY DEVELOPMENT FOR UNMANNED UNTETHERED
SUBMERSIBLE VEHICLE SYSTEMS(U) NEW HAMPSHIRE UNIV
DURHAM MARINE SYSTEMS ENGINEERING LAB JUN 84
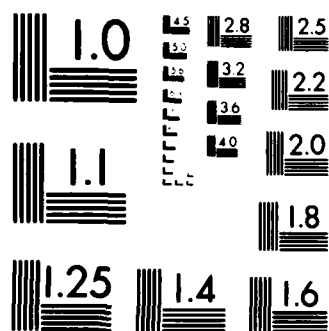4-21292-84-1-P N00014-81-C-0756   F/G 13/10.1 NL

MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963 A

AD-A148 794

TECHNOLOGY DEVELOPMENT FOR UNMANNED

UNTETHERED SUBMERSIBLE VEHICLE SYSTEMS

Final Report

August 1, 1981 - January 31, 1984

Contract #N00014-81-C-0756

Prepared for:

Office of Naval Research
Arctic Program
800 North Quincy Street
Arlington, Virginia 22217

Submitted by:

Marine Systems Engineering Laboratory
University of New Hampshire
Durham, NH 03824

DTIC
ELECTE
DEC 21 1984
S
D
D

June 1984

84 12 11 091

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 4-21292-84-1-P	2. GOVT ACCESSION NO. AD-A148794 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* Technology Development for Unmanned Untethered Submersible Vehicle Systems	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 1 Aug. 1981 - 31 Jan. 1984
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Marine Systems Engineering Laboratory	8. CONTRACT OR GRANT NUMBER(s) N00014-81-C-0756
9. PERFORMING ORGANIZATION NAME AND ADDRESS Marine Systems Engineering Laboratory University of New Hampshire Durham, NH 03824	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-307-468 (461)
11. CONTROLLING OFFICE NAME AND ADDRESS Arctic Program, Arctic & Earth Sciences Division Office of Naval Research 800 North Quincy Street, Arlington, VA 22217	12. REPORT DATE June 1984
	13. NUMBER OF PAGES 225
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* Office of Naval Research Eastern Regional Office 495 Summer Street Boston, MA 02210	15. SECURITY CLASS. *(of this report)* unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

  unclassified document - distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

  N/A

18. SUPPLEMENTARY NOTES

  N/A

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

  unmanned untethered submersible vehicle systems
  Arctic applications

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

  This document describes a program focused on technologies pacing the development of unmanned untethered submersible vehicle systems. Specifically this program has focused on the objective of an Arctic Inspection Mission System (AIMS). This program was to develop and demonstrate the technology required to perform under-ice mapping of the ocean floor as well as the ice keel employing an untethered autonomous submersible. The areas of technology addressed in this regard were:

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73
S/N 0102-014-6601

(1) A sonar sensor array to perform the mapping;
(2) An autonomous untethered submersible vehicle system;
(3) A method of storing the large amounts of data;
(4) Computer requirements to perform this task; and
(5) Definition of a navigation system suitable for the AIMS mission.

PREFACE

This document describes a mulityear development effort focused on the technologies pacing the development of unmanned untethered submersible vehicle systems. Specifically, this program has focused on the potential for using these systems for inspection/work tasks around and within underwater structures. The efforts addressed several technological issues such as developing a computer system capable of accomplishing the Structure Inspection Mission (SIMS), Arctic Inspection Mission System (AIMS), navigation, communication, imaging/bandwidth compression, multibeam sonar imaging, guidance and control. The work described was accomplished under Office of Naval Research Contract Number N00014-18-C-0756 and U. S. Geological Survey Contract Number 14-08-0001-18636.

Accession For		
NTIS GRA&I		X
DTIC TAB		
Unannounced		
Justif...		

A/1

TABLE OF CONTENTS

1. ABSTRACT

The objective of the Arctic Inspection Mission System (AIMS) program (ONR Contract #N00014-81-C-0756) was to develop and demonstrate the technology required to perform under-ice mapping of the ocean floor as well as the ice keel employing an untethered autonomous submersible. The areas of technology addresses in this regard were:

1. A sonar sensor array to perform the mapping.
2. An autonomous untethered submersible vehicle system.
3. A method of storing the large amounts of data.
4. Computer requirements to perform this task.
5. Define a navigation system suitable for the AIMS mission.

2. INTRODUCTION

Over the past ten years, unmanned untethered submersible vehicles have gone from being considered as unusable to a recognition of their importance in many diverse ocean problems. The last year has seen several companies begin development programs directed at utilizing the capabilities of untethered unmanned submersible vehicles. Although the limiting problems associated with this technology are great, the potential rewards resulting from the successful application of these systems seem to compensate for the questionable risk. The Oil and Gas industry, which has acted as the backdrop for the development efforts described in this report, is only one of several application areas which are potentially effected by the advancements in unmanned untethered submersibles.

During the period from November 1, 1980 through February 1, 1984, under Contract Number 14-08-0001-18636 with the U.S. Geological Survey and Contract Number N00014-81-C-0756 with the Office of Naval Research, the Marine Systems Engineering Laboratory has been engaged in the development of technology relating to unmanned untethered underwater vehicle systems. The emphasis of this development effort has been to investigate the technologies which are pacing the development of these vehicles. It has not been to optimize a specific vehicle system or technology, but rather to understand the limiting problems and to provide demonstration of the potential which this technology offers.

The efforts during this period of time have focused on the potential for using unmanned untethered submersible systems for inspection/work tasks around and within an underwater structure. The structure inspection mission system (SIMS) took advantage of the existing EAVE-EAST vehicle system and increased the capabilities of that system in order to autonomously traverse a pre-defined series of paths through an underwater structure.

This report describes a multiyear effort which has focused on several sub-system elements. It is meant to summarize the effort and document the results recognizing both successes as well as problem areas. The work described here is a part of a development effort undertaken by the Marine Systems Engineering Laboratory since 1976. During that period, every effort has been made to understand similar work being accomplished within other research groups in order to eliminate duplication of effort and to utilize the learnings which have resulted from other researchers.

The EAVE program is a cooperative effort between the Marine Systems Engineering Laboratory (MSEL) and the Naval Ocean Systems Center (NOSC), San Diego, to investigate the technologies associated with unmanned untethered sumbersible vehicles. The direction of the work described in this report resulted from a formalized Technology Development Plan developed cooperatively between MSEL and NOSC, San Diego. This effort has hopefully insured maximum utilization of limited funding and resources.

3. EAVE VEHICLE SYSTEM ELECTRONICS

3.1 General System Interrelationships and Operation

The EAVE vehicle is shown in Figure 1. A block diagram of the computer subsystem interaction is shown in Figure 2. The system architecture is one of distributive processors, each of which has its own functions to perform (Tables 1, 2, 3, 4). Many of these functions are performed simultaneously. The critical computer which makes major decisions and controls the overall vehicle performance is the 68000 command computer. The command computer communicates with all of the vehicle sensors and computer systems.

The general timing for the system is shown in Figure 3. A cycle time for the system is the time it takes for the 68000 command computer to acquire all the data from sensors and computers, perform its calculations, make its decisions regarding where it is, where it wants to go and how to get there, and send its commands. The present system operates at a cycle time of approximately 1.5 seconds.

During one cycle period the command computer reads pressure to determine depth (z), reads the compass to determine bearing, reads the navigation computer to acquire new range data and the vehicle x, y position. It then translates this position to the center of the vehicle, and computes changes in bearing (θ) x, y, z, and time (t).

The 68000 computer then looks at its command list for the particular mission. It determines where it is going compared to where it is and issues appropriate instructions to the thruster computer to perform the maneuvers necessary to keep it on course. After the commands are issued the 68000 computer continues to repeat these functions.

At the end of every third cycle the command computer sends 128 bytes of data to the magnetic bubble memory computer for storage. The data consists of all essential information gathered during the three previous cycles as well as the instructions sent by the command computer. This data makes possible a complete mission analysis after the fact, and allows the operator to replot the vehicle path and compare vehicle commands to actual vehicle performance.

The magnetic bubble memory system (MBM) is a nonvolatile high density data storage system. The present configuration has a 1 megabit data storage capacity. A 6100 computer controls the MBM device, receives and/or transmits data and provides the file management for the data. The file management structure is described in Section 11. For the Structural Inspection Mission System (SIMS) mission the data is stored in time sequence such that the first data word of each data cycle is the actual mission time. For each data cycle (command computer cycle time) the data
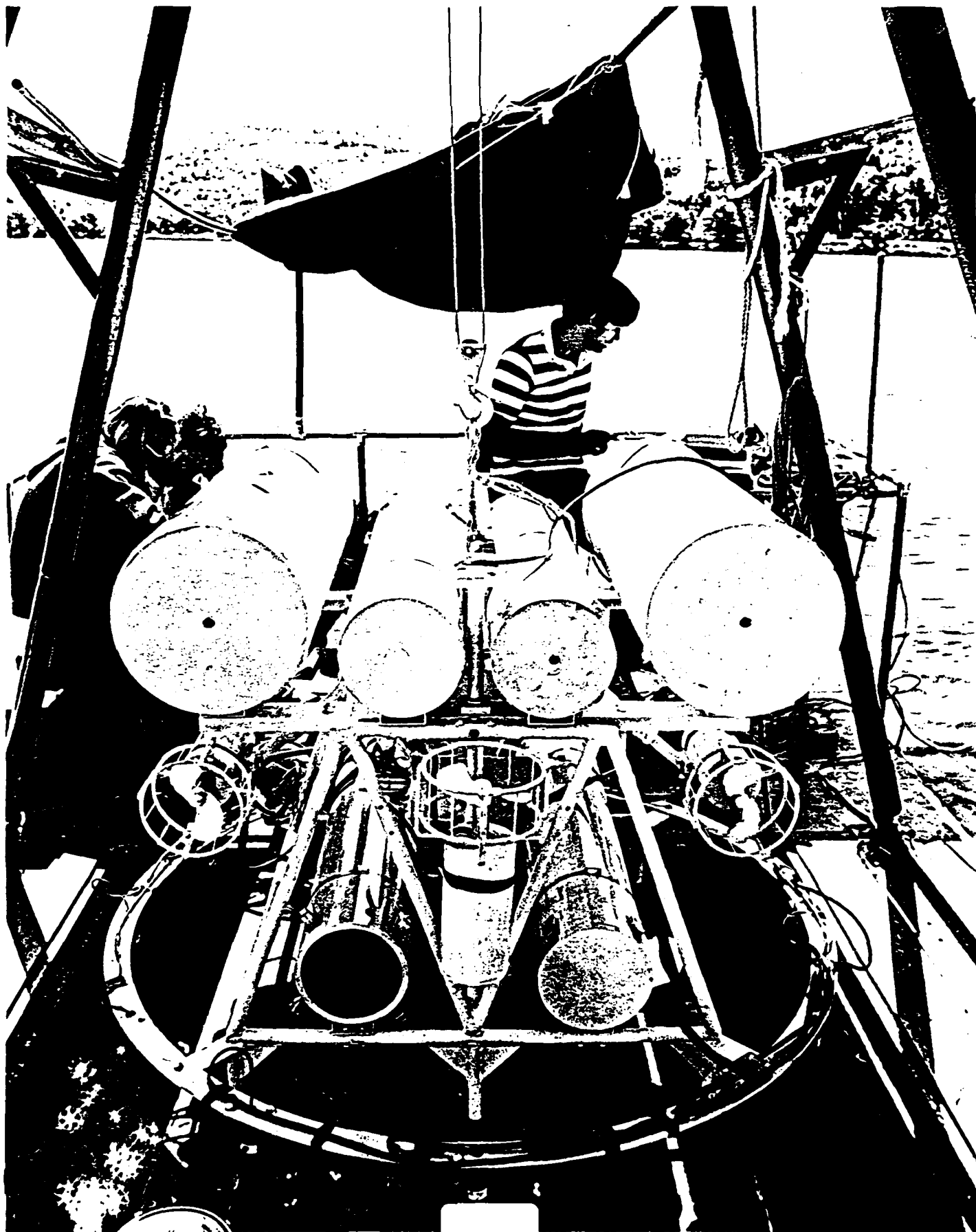
TABLE 1

68000 COMMAND COMPUTER FUNCTIONS

- Maintains time clock.

- Reads pressure and calculates depth (z).

- Reads compass and computes relative bearing (O)

- Requests x, y position from navigation computer and translates it to center of vehicle.

- Computes changes in: time, x, y, z, and O.

- Decides how to control thruster motors in order perform a list of movement commands (such as SIMS mission).

- Passes data to magnetic bubble memory for stora

TABLE 2

NAVIGATION COMPUTER SYSTEM FUNCTIONS

- Keys 95kHz transmitter on vehicle.

- Receives and counts total time to each transponder and back (9 twelve bit words).

- Requests and receives vehicle depth (z) from command computer.

- Calculates x, y position of transmitter on vehicle in its math processor.

- Upon request passes all data above to command computer.

TABLE 3

MAGNETIC BUBBLE MEMORY SYSTEM FUNCTIONS

- Stores mission data from command computer upon request.

- 1.024 Mbit data storage capcity and an additional 2% for file overhead.

- Maintains file system which contains 20 files each with 50 block capacity. Each block contains 128 bytes. Files can be addressed in random access.

- Allows post mission analysis (i.e. reconstruction of mission parameters).

TABLE 4

THRUSTER COMPUTER SYSTEM FUNCTIONS

- Turns on vehicle thrusters in proper direction and at proper speeds upon request from command computer.

BLOCK DIAGRAM OF VEHICLE SYSTEMS

FIGURE 2

EAVE VEHICLE PROCESSOR TIMING
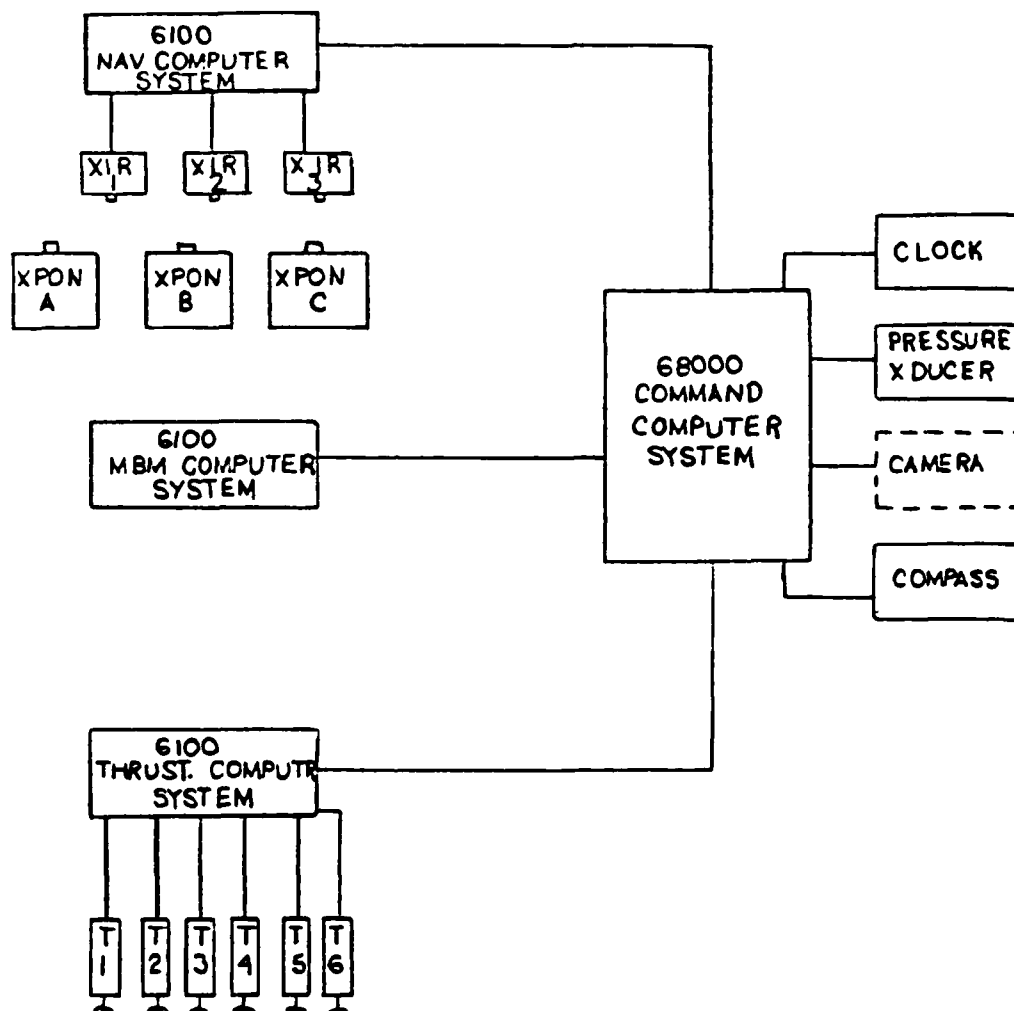
TIME	CYCLE 1						CYCLE 2	CYCLE 3
COMMAND COMPUTER	READ PRESSURE	READ COMPASS	READ NAV	TRANSLATE X Y Z	COMPUTE Δθ ΔX ΔY ΔZ Δ+	DECIDE BASED ON MISSION / COMMAND THRUSTER COMPUTER	REPEAT	REPEAT & STORE ALL DATA FOR 3 CYCLES IN MBM
NAV COMPUTER	X MIT, RECEIVE CALCULATE X, Y			REPEAT			REPEAT	REPEAT	REPEAT	REPEAT
THRUSTER COMPUTER	TURN ON SELECTED MOTORS AT PROPER SPEEDS IN PROPER DIRECTION AS DETERMINED BY COMMAND COMPUTER.						REPEAT	REPEAT
COMPASS	READ BEARING	REPEAT	10 PER CYCLE				REPEAT	REPEAT
PRESSURE	READ PRESSURE	REPEAT	10,000 PER CYCLE				REPEAT	REPEAT
MBM	SLEEP ——————————————————————————————————→							STORE DATA IN MBM

FIGURE 3

stored is that shown in Table 5. Whenever the MBM is not active the 6100 computer puts the MBM to sleep to conserve power.

The navigation computer has a cycle rate which can be varied in software. For the SIMS mission it operates at a speed of 0.4 seconds. During the navigation cycle the navigation system selects a transmitter, sends out an interrogate pulse at 95kHz and receives nine return signals, three from each transponder (see typical transponder configuration in Figure 4). It then determines which data to use, acquires a depth (z) from the command computer, and uses its math processor to calculate an x, y position. It then continuously repeats this cycle. Whenever the command computer interrupts the navigation computer the navigation computer sends the nine return signal counts, the calculated x, y position and a reliability word.

The function of the thruster computer is to control the six thruster motors on the vehicle. The thruster computer can address any or all thrusters and select any one of 31 speeds in either the forward or reverse thrust direction for each thruster. The thrusters are oriented on the vehicle as shown in Figure 5.

The thruster computer makes no decisions, it merely carries out the commands sent to it by the 68000 command computer.

3.2 Overall Vehicle System Software

The software required (see software block diagram Figure 6) to make the subsystems perform interactively consists of a series of device handlers, individual computer operating systems (i.e. navigation, MBM, thruster), a vehicle operating system and an assortment of routines directly related to vehicle control for particular vehicle missions (i.e. mission scenarios, control dynamics, etc.).

The individual computer operating systems (navigation, MBM, thruster) are the software which allows each computer system to stand alone in terms of performing its own specific task. For example, the navigation computer software allows the 6100 computer to:

1. Control timing for the transmitters and receivers.
2. Read the counts for range measurement.
3. Assign a weighted value of "goodness" to these counts.
4. Decide which counts to use in calculating an x, y position.
5. Manipulate the math processor to calculate the x, y position.
6. Store the required information in memory.
7. Transfer the data to the command computer upon request.

The individual operating systems for these computers (6100) are written in assembly language, and will in all probability not change very much if at all. The exception is that a new 68000

TABLE 5

DATA FORMAT FOR ONE COMMAND COMPUTER CYCLE

# of Bytes	Desig.	Information Description
2	t	mission time in hundredth of a second
2	x	x position calc. from nav. (counts)
2	y	y position calc. from nav. (counts)
2	z	depth calc. from presure (binary)
2	xlatx	x position translated to center vehicle (binary)
2	xlaty	y position translated to center vehicle (binary)
2	xlatz	z position translated to pinger depth (binary)
2	O	bearing in degrees
2	rel word	xmiter, rec for calc., calc yes/no, past history
(18)	RDM	raw data matrix (counts
(6)	thruster	thruster polarity, speed, for each thruster
42		
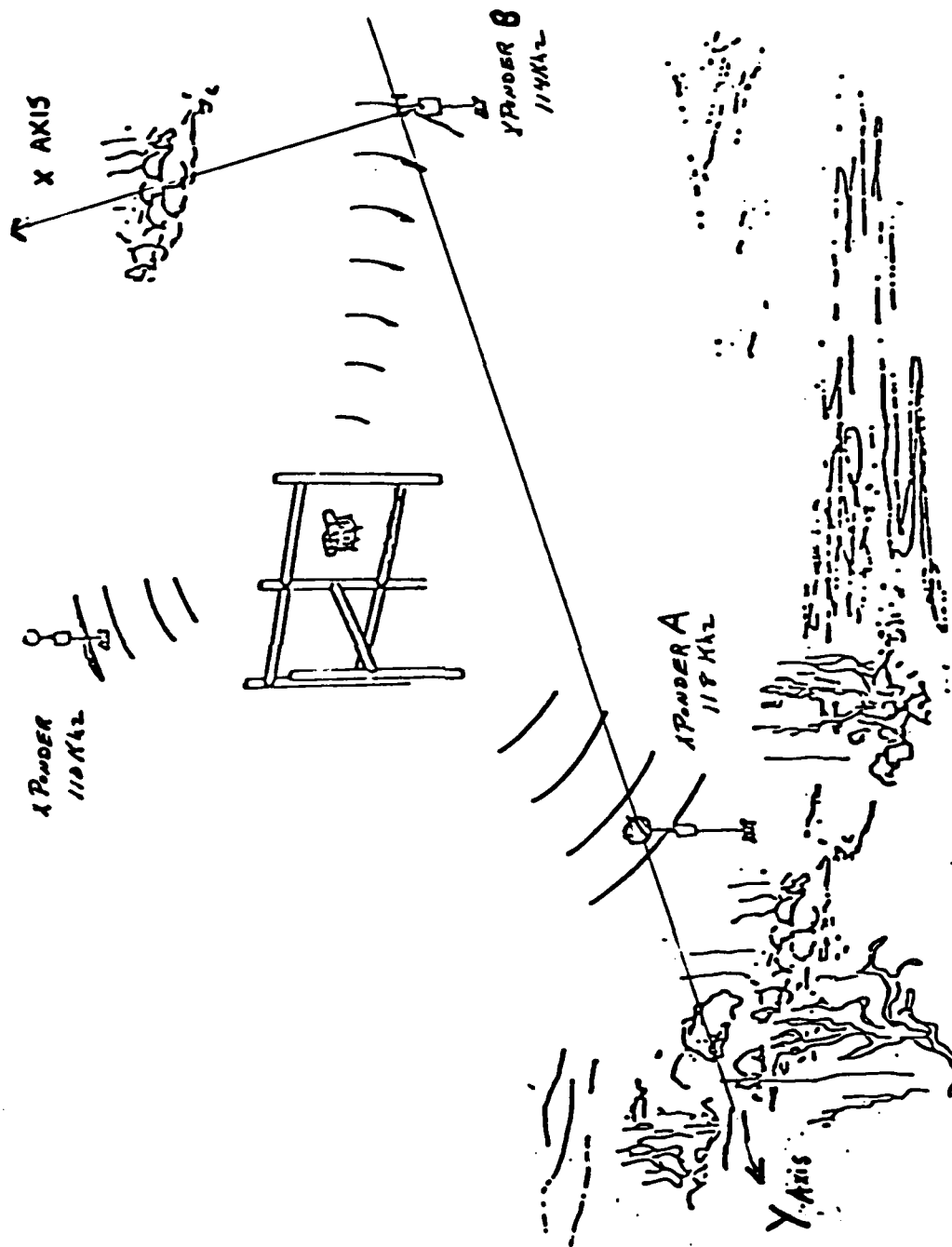
FIGURE 4: TYPICAL TRANSPONDER CONFIGURATION

13

FORWARD

STARBOARD

TOP VIEW
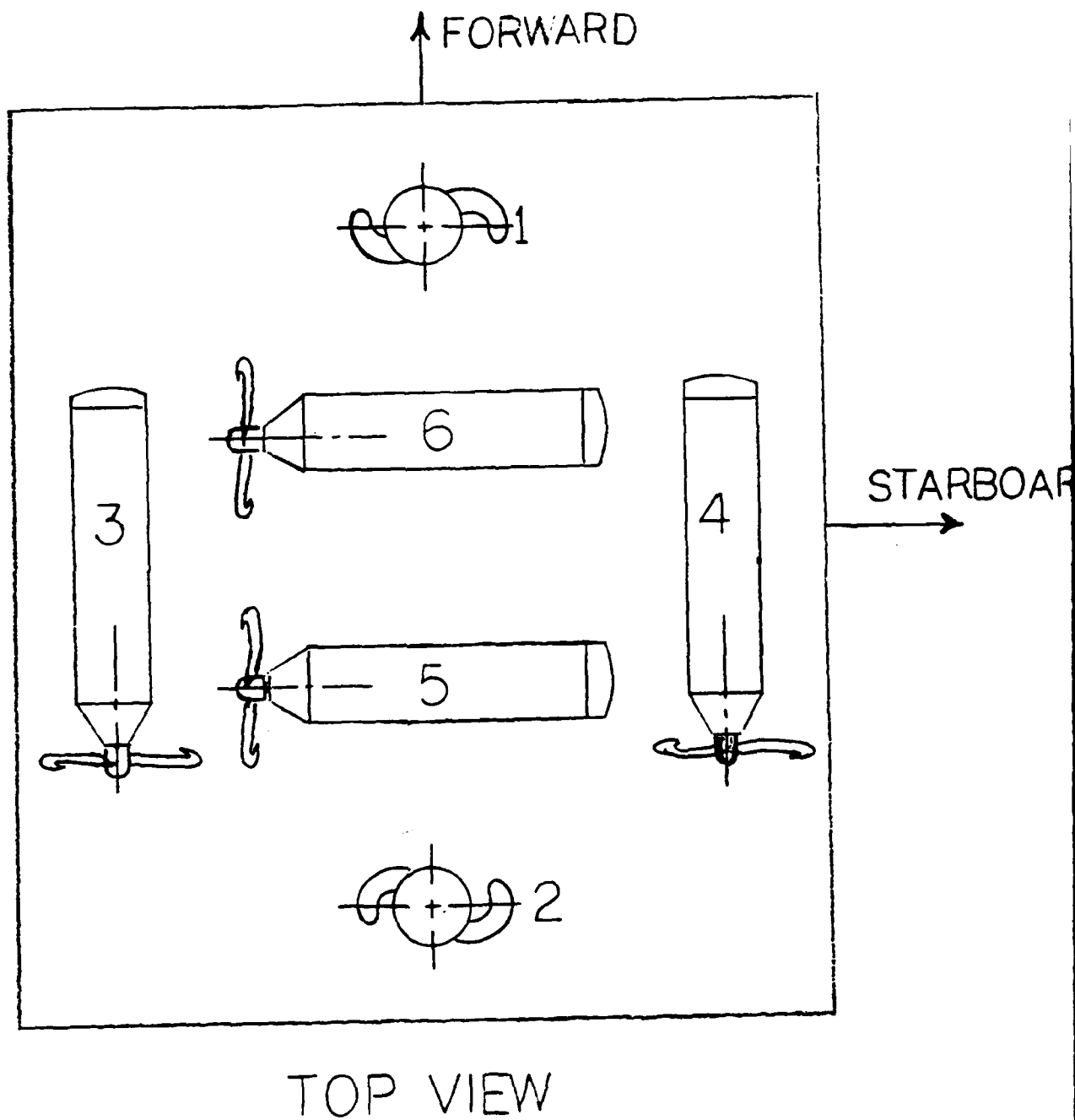
FIGURE 5:   THRUSTER MOTOR ORIENTATION ON EAVE VEHICLE

14

VEHICLE SOFTWARE DIAGRAM
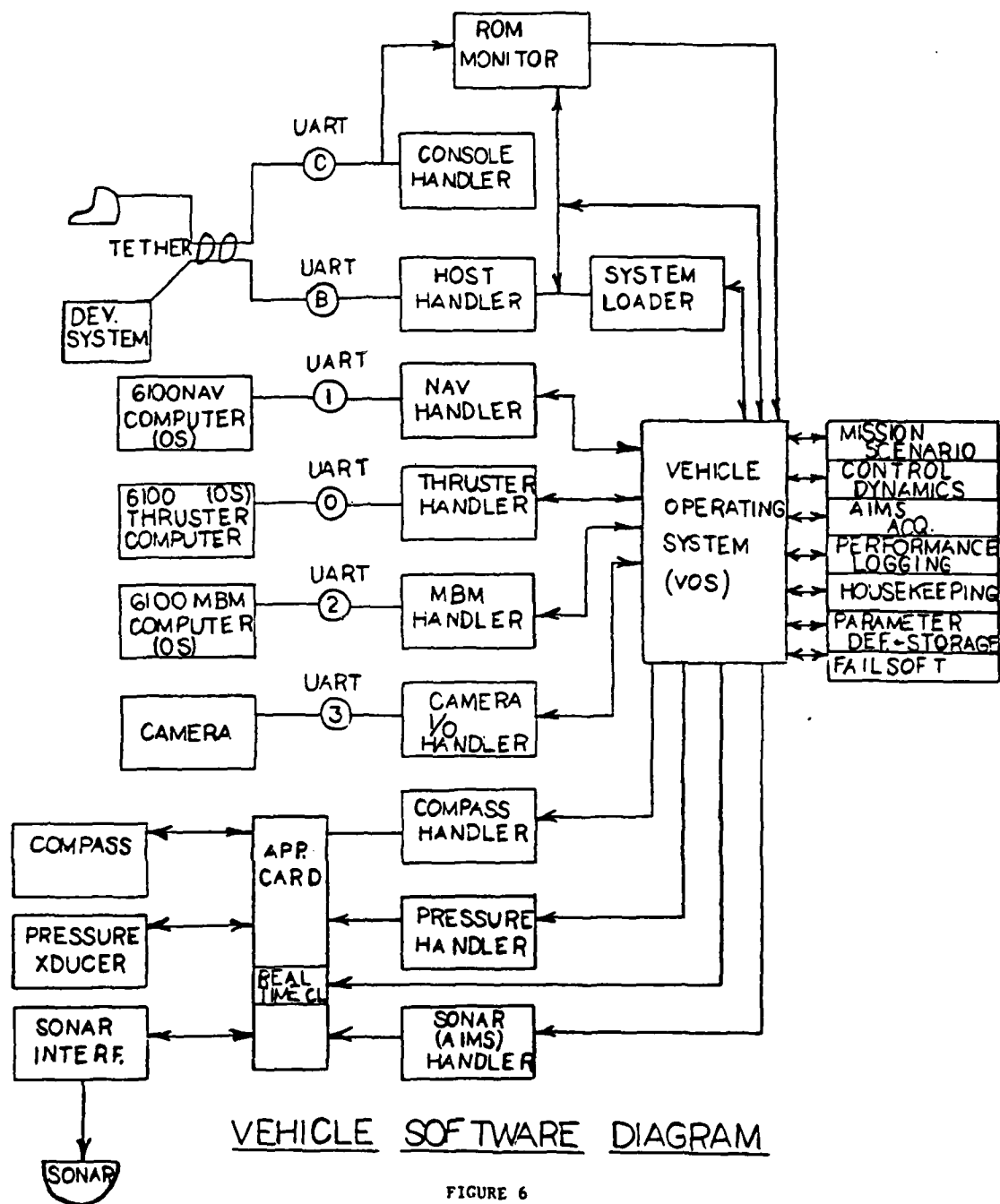
FIGURE 6

based navigation system is being designed for long range navigation and will be programmable in high level language.

There are several "handler" routines written to communicate with each vehicle subsystem. The handlers are really the software interface between each vehicle component and the vehicle operating system. These handlers are all written in high level "C" language.

The vehicle performance programs include:

1. A routine which performs dynamic feedback control functions for the vehicle system.
2. The mission scenario strategy.
3. The AIMS sonar acquisition routine.
4. Data logging for storage in the MBM.
5. Housekeeping.
6. Parameter definition and storage.
7. Fail soft routines.

All of these programs are written in high level language "C". The sections to be changed for different missions would be the mission scenario strategy and the parameter definition and storage routines.

The central nervous system of the vehicle is the vehicle operating system (VOS). It serves to integrate all the vehicle subsystems into a unified whole vehicle. It is the hub of the vehicle system and is also written in high level language. Details of this software are described in Section 7.

3.3 Summary of Electronic System Results

Tests were conducted at Lake Winnipesaukee, N.H. from May through November in 1982 and at Mendum's Pond in Barrington, N.H. from May to November in 1983. Tests were initially performed on subsystems (such as navigation system components, thruster system components, etc.). Both hardware and software components were tested and debugged, and gradually the subsystems were interconnected and tested interactively until the complete vehicle system was assembled and tested as a unit.

Details of subsystem performance are described in each of the subsystem sections of this report. This section presents a brief executive summary of results of the vehicle systems as of this time (November 1983).

A. The thruster computer and thrusters operate reliably. The vertical thruster pair was fitted with a right hand and a left hand prop in order to reduce a tendency for the vehicle to rotate due to precession when both props were of the same type. The results were very dramatic and generated significant savings in power previously required to maintain vehicle heading.

16

B. The Navigation computer system provides for position accuracy of 8 inches for the SIMS type geometry and maintains an x, y position over time to within 4 inches.

A position jitter or shift occurred when switching between transmitters on the vehicle. This is attributable to the transponders not being exactly at the positions provided to the computer. This jitter should be corrected when the self calibration algorithm, described in section 15.6 of this report, is perfected.

Another error occurred on occasion. Due to the variable window strategy use in the navigation algorithm, it is possible for the system to occasionally read a count due to multipath as a good count and hence calculate an erroneous position. In order to discriminate against this occasional gross error in the navigation computer, a simple predictor algorithm was written in "C" for the command computer which recognizes gross position jumps as physically impossible. It does not act on them and requests a new calculation from the navigation computer.

C. The 68000 command computer has performed very reliably over the past year. It is currently operating at a speed of 1.5 seconds per cycle (i.e. complete system update and command execution). This speed should be further reduced to approximately 1 second per cycle once the navigation system has been set up as a separate task function. The thruster computer and magnetic bubble memory computer are already running as separate tasks. This tasking is presently underway and should be completed by mid-December.

Use of the 68000 computer has allowed the writing of sophisticated and complex programs in "C" language in a very short time. Program debugging and changes are very easily managed.

D. The magnetic bubble memory system has also been functioning very well. It has been used consistently to acquire field data on untethered missions and has allowed us to completely reproduce and analyze vehicle performance in the laboratory after each mission.

A plotting program was written and is used in data analysis which allows us to plot 1 megabit of data in approximately 1 1/2 hours (see Figure 11 for example of plot). A complete listing of data is also made for every mission. A minor program bug was found in the 68000 software which communicates with the MBM computer. This is currently being corrected.

E. The final phase of system testing prior to actually sending the vehicle on a complex mission consisted of preprogramming the vehicle to perform certain maneuvers while recording vehicle behavior (in the form of data). The purpose of these

17

tests was to determine the proper gain constant settings in the software feedback control loops. The control algorithm (residing in the 68000 computer in "C" language) is the heart of the vehicle maneuvering system for any type of mission.

Once the proper gain constants were determined the vehicle characteristics were as follows:

	Overshoot (maximum)	Maneuvering	Station Keeping
Depth Accuracy (z)	1.5 ft.	±3 in.	±3 in.
Position Accuracy (x,y)	1.5 ft.	±12 in.	±5 in.
Bearing Accuracy (0)	20 degrees	±15 degrees	±10 deg.

F. Lastly the structure inspection mission was performed. This mission involves a complex set of preprogrammed maneuvers. These maneuvers were all performed with the vehicle within two feet of a structure. A complete description of the mission and plots of vehicle maneuvers are presented in detail in Section 5.

In general, the vehicle performed extremely well. The system performed in a very controlled and stable manner. The missions were varied slightly and ran from 16 minutes to 24 minutes. The vehicle was launched from a 6' x 6' hole in a barge and return to the hole completely autonomously.

A 16mm movie and video tape were made of the vehicle maneuvering in and around the structure.

The vehicle was sent out on 18 structure inspection missions and completed 15 perfectly. On two of the missions that it did not complete, a failsafe mechanism shut the vehicle down due to loss of navigation data. Analysis revealed that one path segment chosen for the vehicle on these two missions was such that the vehicle was very shallow (5 feet deep) and placed the vehicle in a position atop a transponder such that the vehicle was shadowing its own receiver. The cause of the third incomplete mission has not as yet been determined. The symptoms were that the cycle time of the command computer had somehow slowed down to 4.5 seconds instead of its normal 1.5 seconds.

In summary, the EAVE vehicle system is performing as a very well controlled and stable platform. It can repeatedly perform maneuvers and travel underwater over a preprogrammed course and return to a designated location autonomously.

18

4.0 EAVE VEHICLE SYSTEM (MECHANICAL) (Figure 7)

4.1 Space Frame

The EAVE structure frame is of welded 6061-T6 aluminum tubing, 1 in. O.D. x .25 in. wall. Supports for the computer and battery tubes are made from 3 x 1 x 1/8 in. 6061-T6 aluminum channel. Fasteners used on the vehicle are stainless steel. Buoyancy counterweights were made from 1 7/8 in. steel hexagonal stock. There are two lengths used, approximately 33 5/16 in. long each and they were welded together with 3/8 x 1 in. bar stock at each end. The dry weight of this unit is 58.1 lbs. The wet weight is approximately 50.7 lbs. in fresh water. With this amount of ballast the vehicle was buoyant by approximately 10 lbs. The addition of 8 lbs. of lead during field tests gave a satisfactory positive buoyancy estimated at 2 lbs.

An improvement which should be made to the frame is the fabrication of a set of ballast weights of small values (1 & 2 lbs.) which can be easily added or removed from the vehicle to allow the fine tuning of the buoyancy in the field.

The only other item that should be improved is the changing of the starboard forward thruster brace from steel to stainless steel.

4.2 Floatation

The floatation tubes are 12 3/4 in. O.D. x 40 3/4 in. lg. The tubes are of three-piece welded construction, consisting of; a cylinder and a torispherical head at each end. The cylinder is 12 in. sch. 40 round 6061-T6 aluminum pipe with a 12.75 in. O.D. and .406 in. wall and approximately 35 3/4 in. long. The torispherical end caps were formed by McCabe fabricators of Lawrence, Massachusetts from .188 in. thick 6061-T6 aluminum disks. The units were seam welded by Dover Machine Shop, Dover, NH.

A stress analysis of the torispherical dome under external pressure yielded a crush pressure of 638 PSI or a depth of 1473 feet of water. Calculations also show that an aluminum cylinder of the above specified dimensions have a collapse depth of 1400 feet.

Each of the buoyancy tubes are held in place by two 1 in.-8 UNC bolts. Each weighs 57 lbs and displaces 160.2 lbs of fresh water for a net buoyancy of 103.2 pounds. In August 1981 the tubes were updated by replacing the flat ends on the buoyancy tubes with torispherical ends. The update reduced the dry weight, increased the displacement, and reduced forward and reverse drag on the vehicle. The torispherical ends were chosen as a compromise from a spherical or parabolic ends, which would provide the optimum in strength and drag reduction. The torispherical ends were readily available as a standard size

Buoyancy tanks

Acoustic transmitters/receivers

Computers

Compass

Buoyancy tank

Buoyancy tank

Batteries
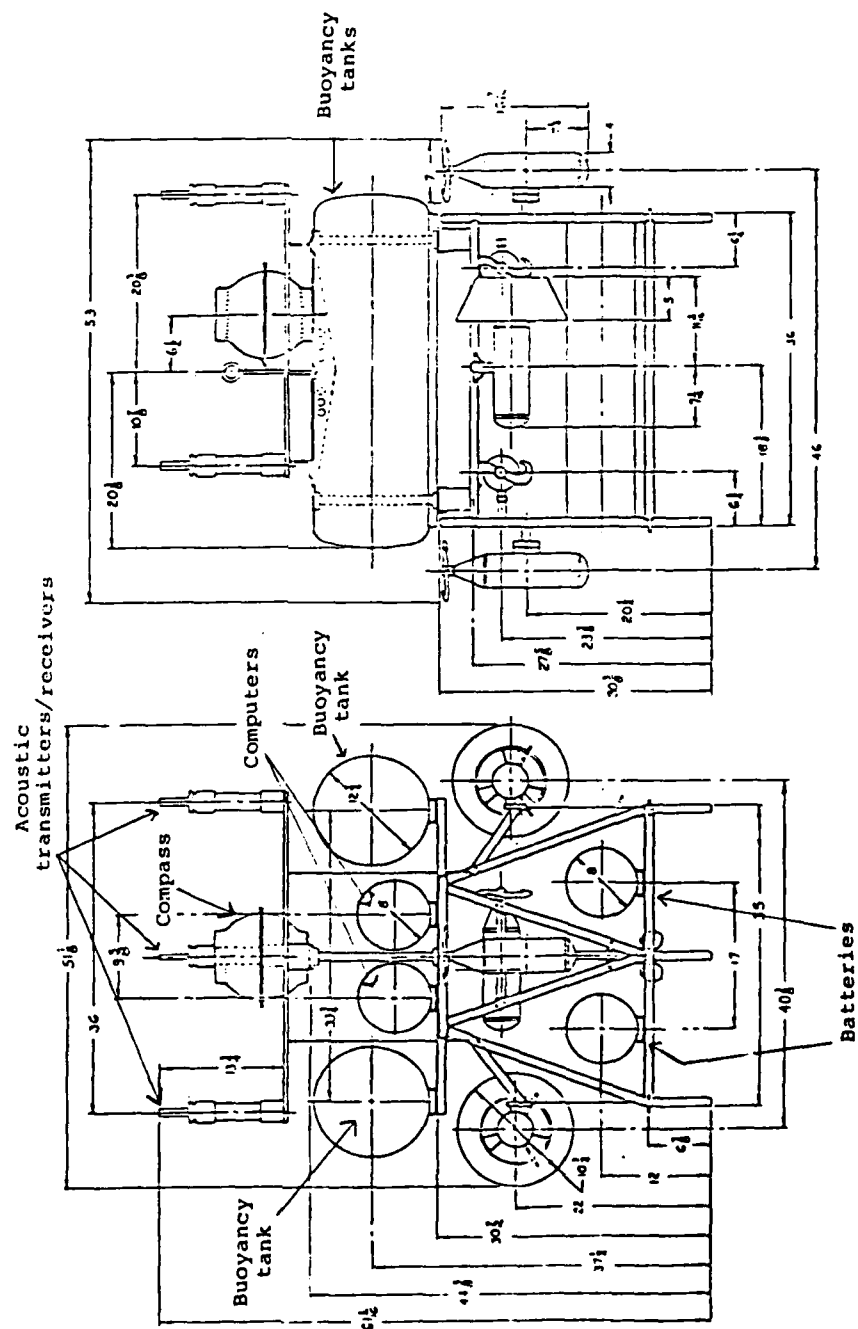
press-formed tank head, and presented themselves as the most economical alternative while providing a strength equivalent to the body of the tube, with a substantial reduction in drag.

The weight reduction of the tubes with the elimination of the flat heads was 13.2 lbs. and the increase in displacement was .31 ft.^{3} or 19.4 lbs of fresh water, producing a net gain of 32.6 lbs per tube and 65.2 lbs total for the vehicle. The increase in buoyancy is counterbalanced with removable weights located at the bottom of the vehicle frame. The excess is intended to allow for the addition of components to the vehicle such as additional batteries or mission related devices.

Calculations show that a 74% reduction in drag was realized through the addition of torispherical ends over the blunt ended tubes used in the past. The magnitude of the force is about .7 lbs. force each at a velocity of 2 ft. per second as opposed to 2.8 lbs. force each for a blunt ended tube. The pressure drag increases with the square of the velocity. (pressure drag = 1/2 Cd ρ V^2 A). The skin friction drag accounts for a very small percentage of total drag. The coefficient of drag for a torispherical head is approximately .2 as compared to .8 for a blunt ended cylinder.

4.3 Computer and Battery Pressure Cases

The two pressure cases which house the control computers on EAVE are made from 8 in. O.D. x .5 in. wall round extruded 6061-T6 aluminum holobar. The tubes are 36 inches in length and calculations show that the crush depth on this size tube is 10,500 feet. The end caps used on these tubes are 3/4 in. thick 6061-T6 aluminum disks. They have an O ring face seal which is nominally rated at 1500 PSI or a depth of 3465 feet of water. This rating could be increased with the use of back-up rings on the seal but the limit of the vehicle is 1400 feet which is established by the rating of the buoyancy tubes.

The two battery pressure cases are made from 8 in. O.D. x 375 wall round extruded 6061-T6 aluminum holobar. These tubes are 36 inches long and are rated for a crush depth of 5,300 feet but the O ring face seals on its 3/4 inch thick end caps restrict this rating to 3465 feet similar to the computer cases.

All end caps at present have flat ends with sharp corners. Drag in the forward and reverse directions on the vehicle could be greatly reduced if we could provide well rounded corners on these end plates. This could present difficulty in mounting the hold down clips on the cap sides, but if possible, the benefits in power reduction required to drive the vehicle could outweigh the effort required to retrofit.

21

4.4 Thrusters

The thruster motors were manufactured by Minnesota Electric Technology Inc. of Winnebago, Minn. The no load RPM is 1750 when operated at 24VDC. Its peak hp is .22 at 800 RPM drawing about 12.5 amps at that point. The actual load exerted by the propeller on the motor is not known, but under full load the amperage measured during an evaluation of prop Kort Nozzle on the forward thrusters was 12.2. If we relate this amperage to the theoretical performance curve supplied by Minn. Electric it would indicate that we are at the peak hp available and are operating at about 800 RPM. The forward thrusters are equipped with a 7 1/2 in. dia. three blade propeller with a 10 inch pitch. The prop and the Kort Nozzle were bought as parts from a Diver Propulsion Vehicle 01-1002MK-11 manufactured by Farlon Oceanic Industries, 14275 Catalina Street, San Leandro, CA 94577 (415-352-5001). The Kort Nozzle was modified by MSEL to adapt to the thrusters and the outside was filled with a syntactic foam to conform it to a true Kort Nozzle shape (foil.)

The vertical and horizontal thrusters do not have Kort Nozzles and are equipped with a 7 inch, two blade prop with a 5 inch pitch. The prop was purchased from Minn. Kota Inc., 201 N. 17th Street, Moorehead, Minn. 56560 (507-345-4623) under Part No. 03308. The prop was used on an electric trolling motor.

Tests on the thrust of both props were conducted in August 1981. Results of these tests showed that the average peak thrust was about 7 lbs. with the two blade Minn. Kota prop in the forward direction and 4 lbs in reverse. There was a wide discrepancy between thrusters with a low of 5.8 lbs and a high of 10.25 lbs. in the forward direction and a low of 3 lbs and a high of 5.6 in reverse.

A study of a test conducted in February, 1979 on the thrusters using this same prop shows an average thrust of approximately 12 lbs. At that time discrepancies between individual thrusters existed and it was found that by cleaning the motor brushes performance could be improved and more consistent results between thrusters obtained. The proposed reason for poor performance was the fouling of the brushes by the pressure compensating oil bath that they were exposed to. At that time they were using a mineral oil. They changed to a Mobil transformer oil and felt they had better results. We have now changed the oil as of 11/82 to a Dow Corning 200 silicone fluid 5 centistroke as we were experiencing erratic behavior (slow running) of the motors in cold temperatures (40 degrees F). This oil seems to have improved the performance with its lower viscosity at these temperatures. Performance tests have been conducted to determine actual performance with this oil and are presented in Section 4.5. It is proposed that we run an endurance test on a selected motor and determine change in performance with respect to time to determine maintenance intervals if fouling is responsible for changes in performance.

22

The major problems in maintaining the thrusters is the refilling of them with oil. To date there has not been a successful method used in filling without entrapping air either in a pocket within the motor or failure to deairate the oil properly prior to filling. Methods used to date have proven ineffective, and time consuming. The proposed fixture and procedure which we believe will eliminate these problems and allow us to establish a convenient and cost effective maintenance schedule for the thrusters has been designed and fabricated.

Visual inspection of the thrusters during disassembly (11/82) showed that the O ring seals in the thruster housing, although still effective, had been chemically attacked by the transformer oil previously used, causing a swelling and stretching of the seals. The O rings used were Buna-N. Because of time constraints the thrusters were filled with the DC200 silicone oil and the same O rings. Buna-N is not completely inert to silicone oil, but can be used if replaced periodically. A better choice of materials would be Viton, which remains unaffected by silicone oil even at elevated temperatures, at about 4 times the cost of Buna-N. In any case the thrusters, when disassembled next, should be equipped with new O rings, preferably Viton. The pressure compensating diaphragm used appeared to be unaffected by the transformer oil, although it would be harder to determine because it is a Buna-N impregnated fabric and tends to maintain its size.

To summarize, it is recommended we take the following steps to correct problems and improve the EAVE thrusters.

1. Fabricate necessary apparatus to allow correct and cost effective maintenance and filling of thrusters.

2. Performance test of thrusters with new silicone oil.
   a. Effect of brush cleaning
   b. Deterioration of brushes with time-establish maintenance schedule
   c. New propeller performance.

3. Change O rings in thruster to Viton, or replace with new Buna-N and observe deterioration with time-establish change schedule.

4.5 Thruster Motor Tests

A group of tests were conducted on the EAVE vehicle thrusters in order to obtain further information in the following areas:

4.5.1 Thruster Performance

Each thruster was removed from the vehicle and tested for thrust vs. command speed using the vehicle's thruster computer and batteries. Thrust measurements were recorded for each of the

31 possible computer command speeds using an L section and scales. This was done for both forward and reverse thrust directions.

In order to test the effectiveness of the Kort nozzles used on the pair of horizontal thrusters, the #1 thruster was tested with the Kort nozzle and its prop and then with a Minn-Kota prop similar to those used on the thrusters not employing Kort nozzles.

Following these tests, the so-called unmarked (spare) thruster was dismantled and inspected. It was found to have a slight puncture in its diaphragm and a build-up of carbon on its brushes and (armature). This build-up was removed with emery cloth and the diaphragm was replaced. The motor bearings, oil seal, and coupling all appeared to be in good condition. The thruster was then re-assembled and the thrust tests conducted once again.

The data and the curves generated may be found in Appendix A. The following observations were made during the tests:

1. The maximum values of thrust for the reverse direction appears to be on the order of 60-70% of that attainable in the forward direction.

2. There were noticeable fluctuations in thrust for all thrusters although some showed greater tendencies than others. The thrust results, at constant command speed, were seen to vary from 5% to 20%, the exact reasons for the variations are not known at this time. Further investigation into this occurrence is planned.

3. As the testing proceeded, an increasingly noticeable film of oil appeared on the surface of the test tank. The leakage appeared to be occurring predominantly at the diaphragm although the shaft seal is also suspect.

4. While the Kort nozzles did show improvement in thrust at the intermediate command speeds, at maximum speed the difference was negligible. This indicates that, when the drag of the nozzles comes into effect during vehicle operation, during high speed, the nozzles may actually decrease the net maximum achievable thrust.

4.5.2 Effect of Byrd Prop

This test involved replacing one of the right-hand Minn-Kota props on the vehicle's vertical thrusters with a Byrd left-hand prop. This was done in an effort to eliminate the tendency of the vehicle to precess when the vertical thrusters are operating.

The vehicle was first lowered into the test tank with similar Minn-Kota props on both thrusters in question. Using the vehicle's thruster computer, the thrusters were run at various

command speeds. As the speeds were increased, the rotation of the vehicle became quite apparent. The vehicle was then removed from the tank, one of the Minn-Kota props replaced with a Byrd left-hand prop and the test was conducted again. This time none of the precessional tendencies previously observed occurred. This indicates that the Byrd props would serve to correct the unwanted precession which had been experienced in previous vehicle field testing.

Following the successful test of the Byrd prop on the vehicle, the prop was tested for thrust vs. command speed, in the manner previously described, in order to determine its effect on thrust. The test indicated that the maximum attainable thrust using the Byrd prop is approximately 90% of that achievable using the Minn-Kota prop.

4.5.3 Temperature Effects on Thruster Performance

Previous experience had shown that at reduced temperatures, in the 35 - 40 degree F range, the thrusters suffered an extreme loss of shaft speed and therefore performance. This unacceptable phenomenon was attributed to the increasing viscosity of the transformer oil being used for pressure compensation. The oil was therefore replaced with dow corning #200 fluid, a clear silicone fluid which exhibits a relatively flat viscosity slope over the expected operating temperatures.

The test consisted of the thruster being submerged, in a vertical position, up to its shaft seal in a water bath. The vehicle's batteries were used to supply a constant voltage directly to the thruster. The shaft speed (rpm) of the operating thruster was monitored as the temperature was gradually reduced, using ice, down to approximately 1 degree C and then allowed to rise again. Temperature and shaft speed were recorded at one minute intervals. The data shows no loss of shaft speed which can be attributed to the effects of temperature changes. The variations which occurred are believed to be only the random fluctuations observed previously in the performance tests. The lack of temperature related reductions in shaft speed indicates that the Dow Corning fluid will correct the problem previously encountered at reduced temperatures.

4.5.4 Conclusions

The tests conducted have shown that the two problems previously encountered, first, the unwanted precession due to like props on thruster pairs, and second the loss of thruster performance at reduced temperatures have been corrected. Further investigation is yet to be conducted to pinpoint the cause and possibly eliminate the fluctuations in thrust.

5.0 SIMS MISSION

The SIMS mission was designed to demonstrate the ability of an untethered submersible robot to maneuver itself in a very precise manner through a three dimensional structure. In order to perform this task the robot must be capable of determining its own position in space (x, y, z,θ) at all times. It must also carry with it a map of its trajectory through the structure; as well as the location of points within the structure at which it must stop for inspection. It should also be capable of storing all the data necessary to analyze and/or duplicate its behavior for post mission analysis.

The structure used for the SIMS mission is sketched in Figure 8. It is made of 4 in. diameter PVC pipes. The "window" openings are approximately 8 feet by 8 feet. The maximum vehicle dimension is approximately 5 feet, hence it has a clearance of approximately 1 1/2 ft. when it penetrates the structure window.

The path and maneuvers of the SIMS mission can best be understood by referring to Figure 9. The structure is located in the x, y, z plane which is superimposed (in software) on the transponder network. The transponders are not shown in the figure.

The mission begins with the vehicle being placed in the water through a six foot by six foot hole in the barge. After a selectable delay time has elapsed (to allow for disconnecting the tether and placing the vehicle in the water) the vehicle begins its descent. As it does so it reads the first command (see Figure 10) which tells the vehicle to dive to 10 feet, at position x = 5, y = 40 and turn to a heading of 4.9 radians. Once the vehicle has acquired its position at point START to within one foot, its bearing to within six degrees, and the command duration has expired, it reads the next command which is a horizontal move to point A. The system is programmed such that the vehicle will not change commands until all three conditions above are met. Once the vehicle is at point A, it rotates to face the structure window, then dives to position itself in the center of the window (point B). The next command instructs the vehicle to proceed on a straight line forward to position C at the center of the horizontal beam. Next it turns to face the beam. At this point it is two feet from the beam. The next two commands instruct the vehicle to move forward one foot and stay there for 30 seconds and then back up one foot to position C. The next command instructs the vehicle to move horizontally sideways (slide left) while maintaining its bearing and depth until it arrives at point D. At point D the vehicle moves up four feet (to point E) and then back down (point D) to inspect the vertical end beam of the structure. It holds its x, y position and bearing during this maneuver.

The mission proceeds in this fashion as the vehicle slides right to point C and then moves up and over the beam to the other side of the structure. It performs several more maneuvers on
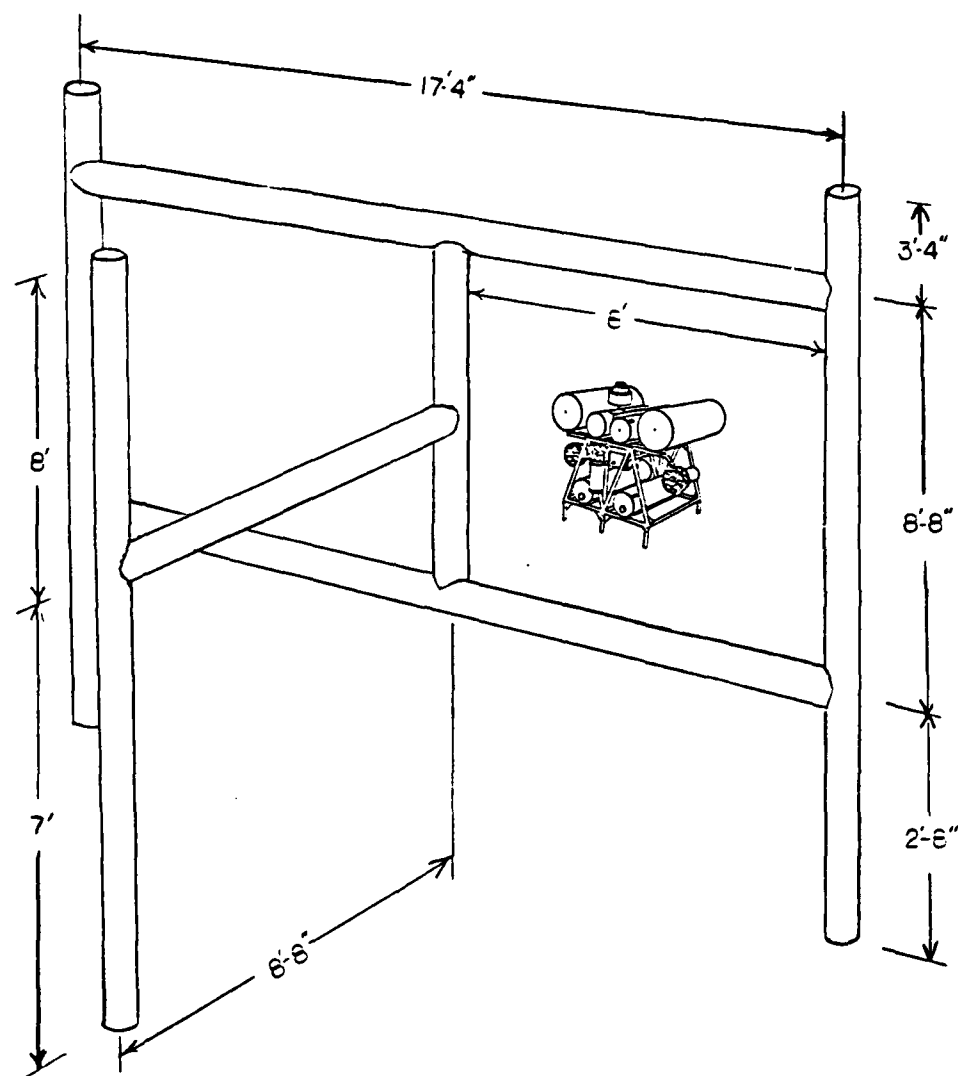
26

FIGURE 8:   TEST STRUCTURE
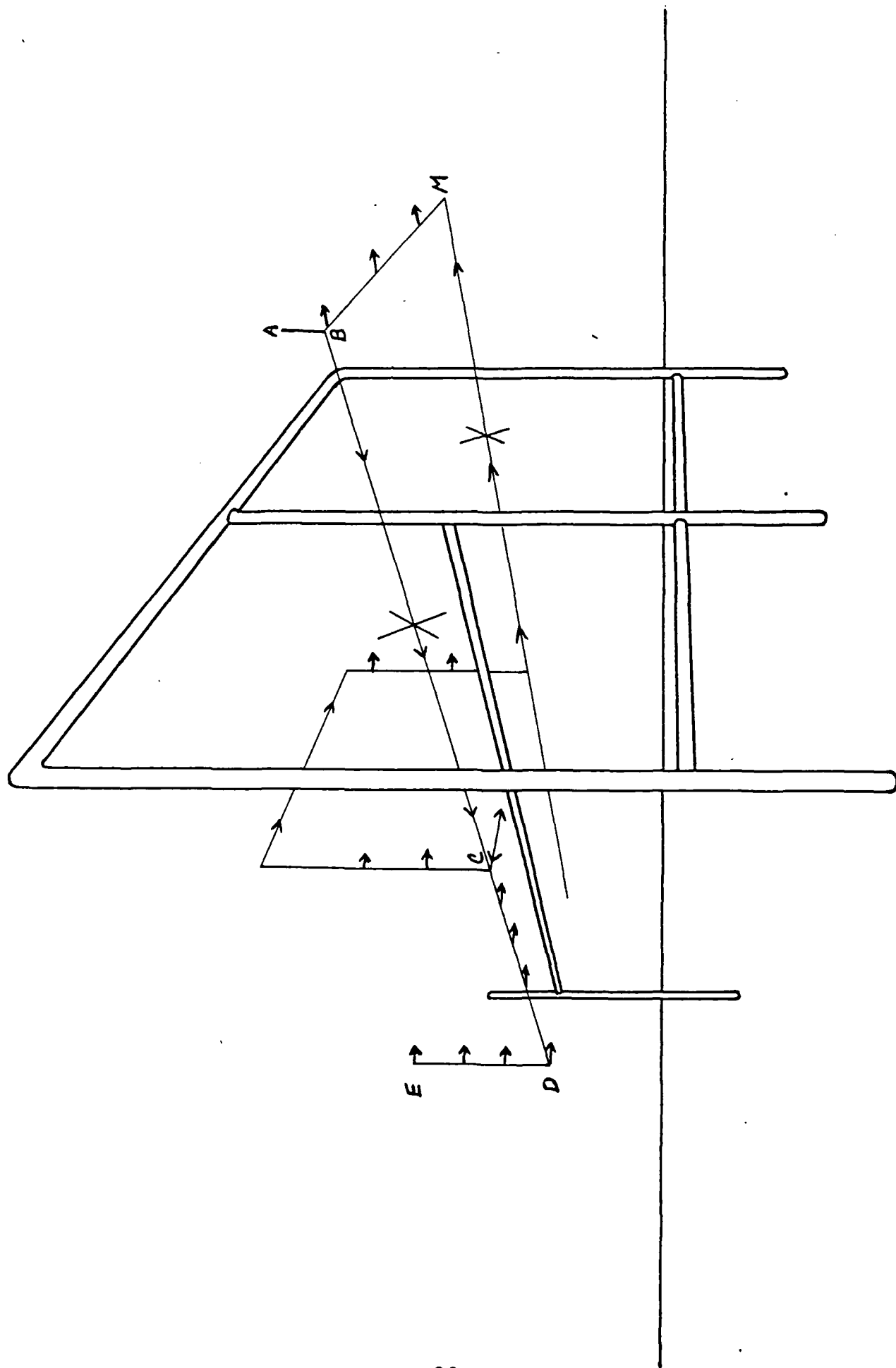
FIGURE 9: MISSION PATH

28

```
/*-----------------------------------------------------------------------*/
/* array of NUMCOMS commands; 0th element is default emergency exit */
COMM_BLOCK   comms[NUMCOMS]{

/*  0 */ EXIT,           FALSE,  100,      1.1,   2.2,   3.3, 4.4, 5.55, 6.666,

/* command type          ignore xy duration   x      y      z    xdot bear  bdot */
      #
/*  1 */ VERT_MOVE,      FALSE,  6000,     5.0,  40.0,  10.0, 1.0, 4.90, 0.225,
/*  2 */ HORIZ_MOVE,     FALSE,  4000,    12.5,  20.6,  10.0, 1.0, 0.00, 0.225,
/*  3 */ ROTATE_ONLY,    FALSE,  6000,    12.5,  20.6,  10.0, 1.0, 0.33, 0.225,
/*  4 */ NOP,            TRUE,   000,      1.1,   2.2,   3.3, 4.4, 5.55, 6.666,
/*  5 */ NOP,            TRUE,   000,      1.1,   2.2,   3.3, 4.4, 5.55, 6.666,
/*  6 */ VERT_MOVE,      FALSE,  6000,    12.5,  20.6,  12.5, 1.0, 0.33, 0.225,
/*  7 */ HORIZ_MOVE,     FALSE,  6000,    23.5,  26.0,  12.5, 1.0, 0.00, 0.225,
/*  8 */ ROTATE_ONLY,    FALSE,  6000,    23.5,  26.0,  12.5, 1.0, 1.90, 0.225,
/*  9 */ HORIZ_MOVE,     FALSE,  3000,    23.0,  27.0,  12.5, 1.0, 0.00, 0.225,
/* 10 */ HORIZ_MOVE,     FALSE,  3000,    23.5,  26.0,  12.5, 1.0, 3.14, 0.225,
/* 11 */ HORIZ_MOVE,     FALSE,  6000,    28.0,  29.5,  12.5, 1.0, 1.57, 0.225,
/* 12 */ VERT_MOVE,      FALSE,  3000,    28.0,  29.5,   8.5, 1.0, 1.90, 0.225,
/* 13 */ VERT_MOVE,      FALSE,  3000,    28.0,  29.5,  12.5, 1.0, 1.90, 0.225,
/* 14 */ HORIZ_MOVE,     FALSE,  6000,    23.5,  26.0,  12.5, 1.0, 4.71, 0.225,
/* 15 */ VERT_MOVE,      FALSE,  4000,    23.5,  26.0,   8.5, 1.0, 1.90, 0.225,
/* 16 */ HORIZ_MOVE,     FALSE,  6000,    19.0,  33.5,   8.5, 1.0, 0.00, 0.225,
/* 17 */ VERT_MOVE,      FALSE,  4000,    19.0,  33.5,  12.0, 1.0, 1.90, 0.225,
/* 18 */ ROTATE_ONLY,    FALSE,  6000,    19.0,  33.5,  12.0, 1.0, 0.33, 0.225,

/* 19 */ HORIZ_MOVE,     FALSE,  6000,    24.0,  35.5,  12.0, 1.0, 0.00, 0.225,
/* 20 */ ROTATE_ONLY,    FALSE,  6000,    24.0,  35.5,  12.0, 1.0, 5.04, 0.225,
/* 21 */ HORIZ_MOVE,     FALSE,  6000,    19.0,  33.5,  12.0, 1.0, 1.57, 0.225,
/* 22 */ ROTATE_ONLY,    FALSE,  6000,    19.0,  33.5,  12.0, 1.0, 3.45, 0.225,
/* 23 */ HORIZ_MOVE,     FALSE,  6000,     8.5,  27.7,  12.0, 1.0, 0.00, 0.225,
/* 24 */ HORIZ_MOVE,     FALSE,  6000,    12.5,  20.6,  12.0, 1.0, 4.71, 0.225,
/* 25 */ VERT_MOVE,      FALSE,  4000,    12.5,  20.6,  10.0, 1.0, 3.45, 0.225,
/* 26 */ ROTATE_ONLY,    FALSE,  6000,    12.5,  20.6,  10.0, 1.0, 2.00, 0.225,
/* 27 */ HORIZ_MOVE,     FALSE,  6000,     5.0,  40.0,  10.0, 1.0, 0.00, 0.225,
/* 28 */ VERT_MOVE,      FALSE,  3000,     5.0,  40.0,   6.0, 1.0, 2.00, 0.225,
/* 29 */ EXIT,           FALSE,  100,      1.1,   2.2,   3.3, 4.4, 5.55, 6.666,

/* 30 */ EXIT,           FALSE,  100,      1.1,   2.2,   3.3, 4.4, 5.55, 6.666,
         };
```

FIGURE 10:   COMMAND LIST FOR INSPECTION MISSION

29

that side of the structure and finally exits the right hand window to point M. Next it slides to point B, its original starting point, and it climbs to 10 feet and turns to head back to the barge.

Once it gets to its position below the hole in the barge, it climbs to six feet, and once there shuts off power and floats up into the hole in the barge.

Figure 11 is a plot of vehicle behavior for the mission described above. This plot was made from data stored in the magnetic bubble memory during the mission. A record is made and is plotted for every command computer cycle (i.e. 1.5 second intervals). The vehicle position in x, y, and z and the vehicle bearing are all plotted against time. The straight lines drawn on the plot represent the idealized perfect response. The vertical dotted lines on the plot indicate a command change. A brief description of each command is printed in the execution time period for that command.

It is very easy to visualize the system response and stability if one looks at the plots carefully. These plots indicate very clearly that the EAVE vehicle is very stable. The vehicle can hold on station within ±1 foot of desired position and within ±6 degrees. Most of the time it is even better than this figure (±.5 feet). (See for example sections 3.1 and 6.1 on page 2 of Figure 11.

The plots also indicate the system responses in terms of overshoot and damping and are self evident.

FIGURE 11: November 9, 1983
Structure Inspection Mission

Dive to 10 ft.
acquire position
x = 5, y = 40,
θ = 280°

LEGEND

x = x position in feet
y = y position in feet
H = depth in feet
θ = bearing in degrees
= ideal mission path

= command change
| = 10 command computer cycles = 15 seconds

Note: data starts 7 seconds after initial thruster downburst
in order to be deep enough to acquire navigation data.

FEET

0   6   12   18   24   30   36   42   48   54

DEGREES

0   60   120   180   240   300   360
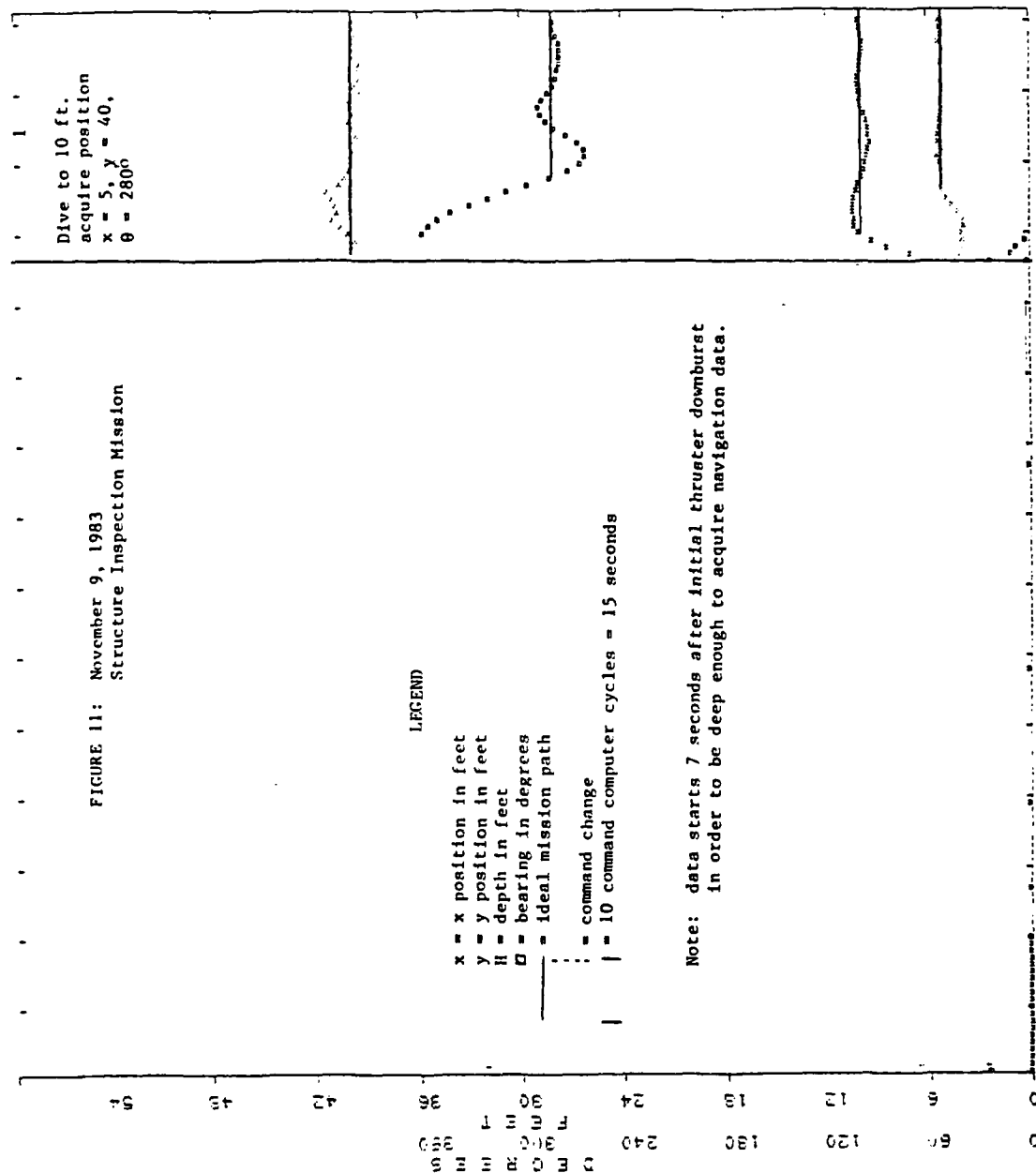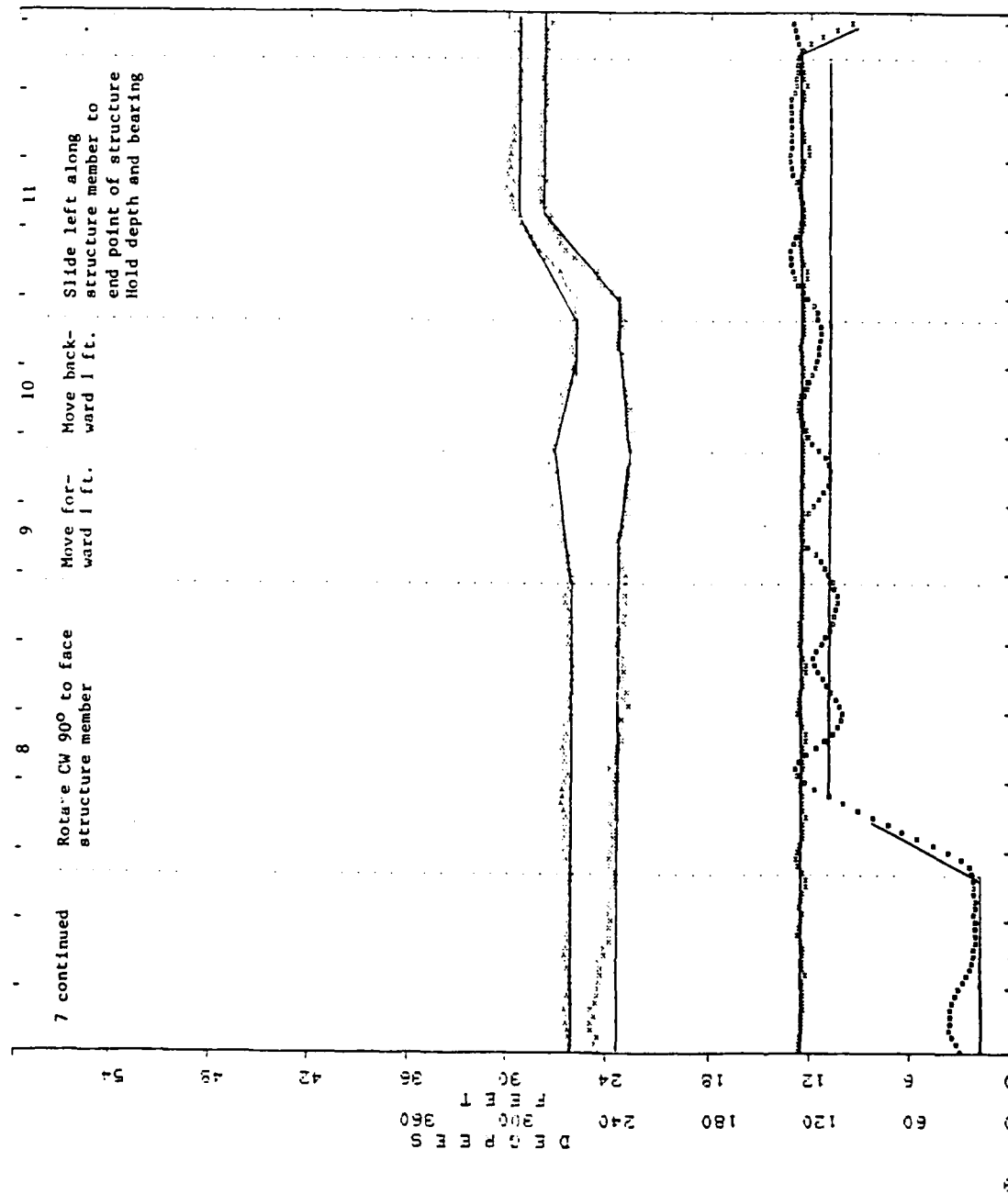
Figure 11 continued

Figure 11 continued

33

12	13	14	15	16
Climb 4 ft. to (Inspect vertical structure member Hold x, y, θ	Dive back down 4 ft. Hold x, y, θ	Slide right to center of structure member (point 4) Hold z, θ	Climb to 8.5 ft. Hold x, y, θ	Move forward over structure member to other side

5'
48
42
36
30
24
18
12

F E E T

360 300 240 160 20

D E G R E E S
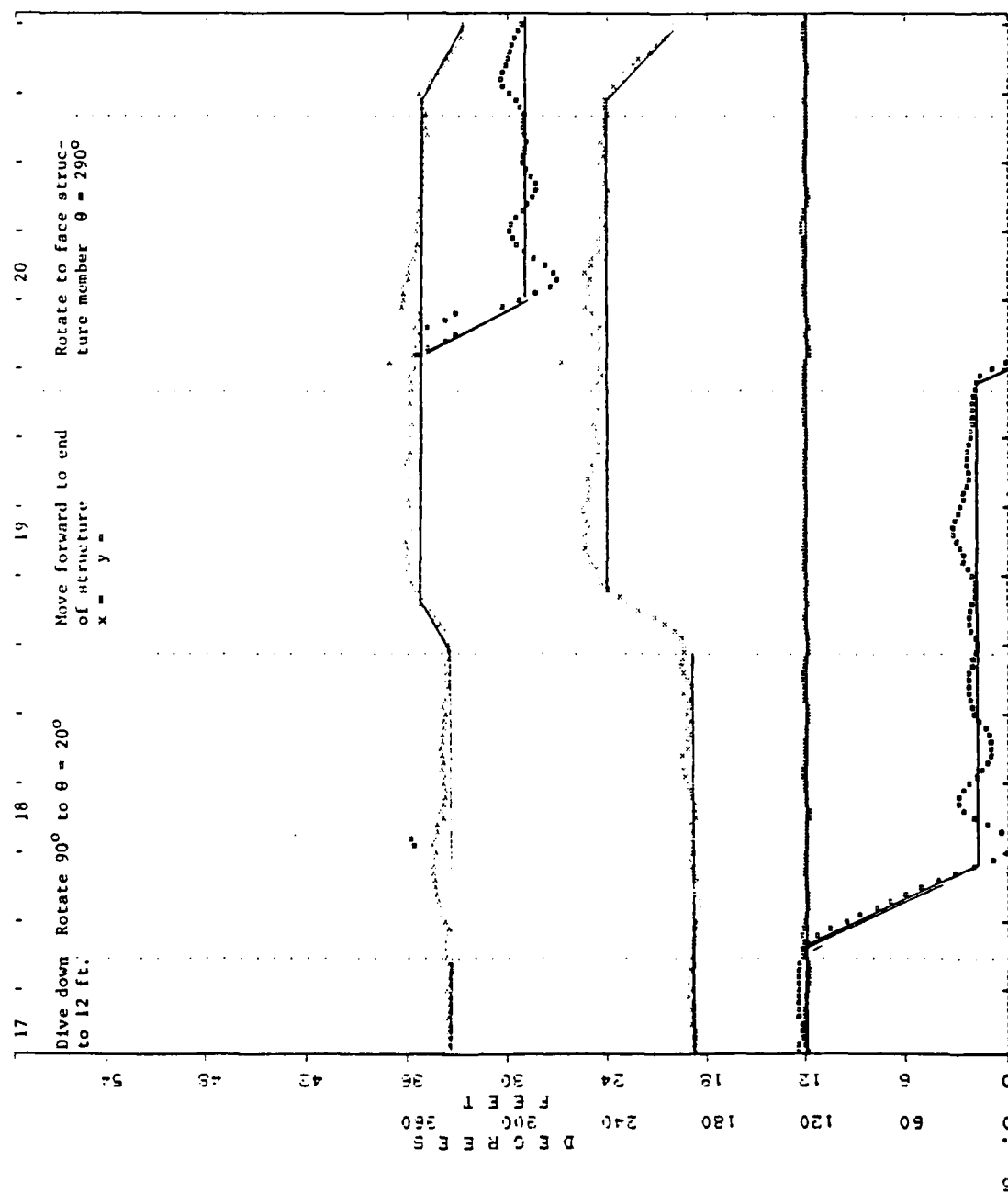
Figure 11 continued

6. 68000 COMMAND COMPUTER SYSTEM

6.1 Overview (Figure 12)

A versatile autonomous robot such as the SIMS vehicle requires a reasonable amount of computer capability to easily perform complex motion control and image processing. For optimum flexibility, the on-board computer system should support high-level languages so that software can easily be documented and adapted to changing system needs. In addition, the on-board computers must use power efficiently so that they can operate for fairly long periods of time from the vehicle battery packs.

The Motorola 68000 microcomputer was selected for the main computer system because of its efficient architecture and the available UNIX and C software support. This microcomputer was then surrounded with modern high-speed CMOS parts to minimize power consumption. A bus architecture was devised to permit complete flexibility in future expansion.
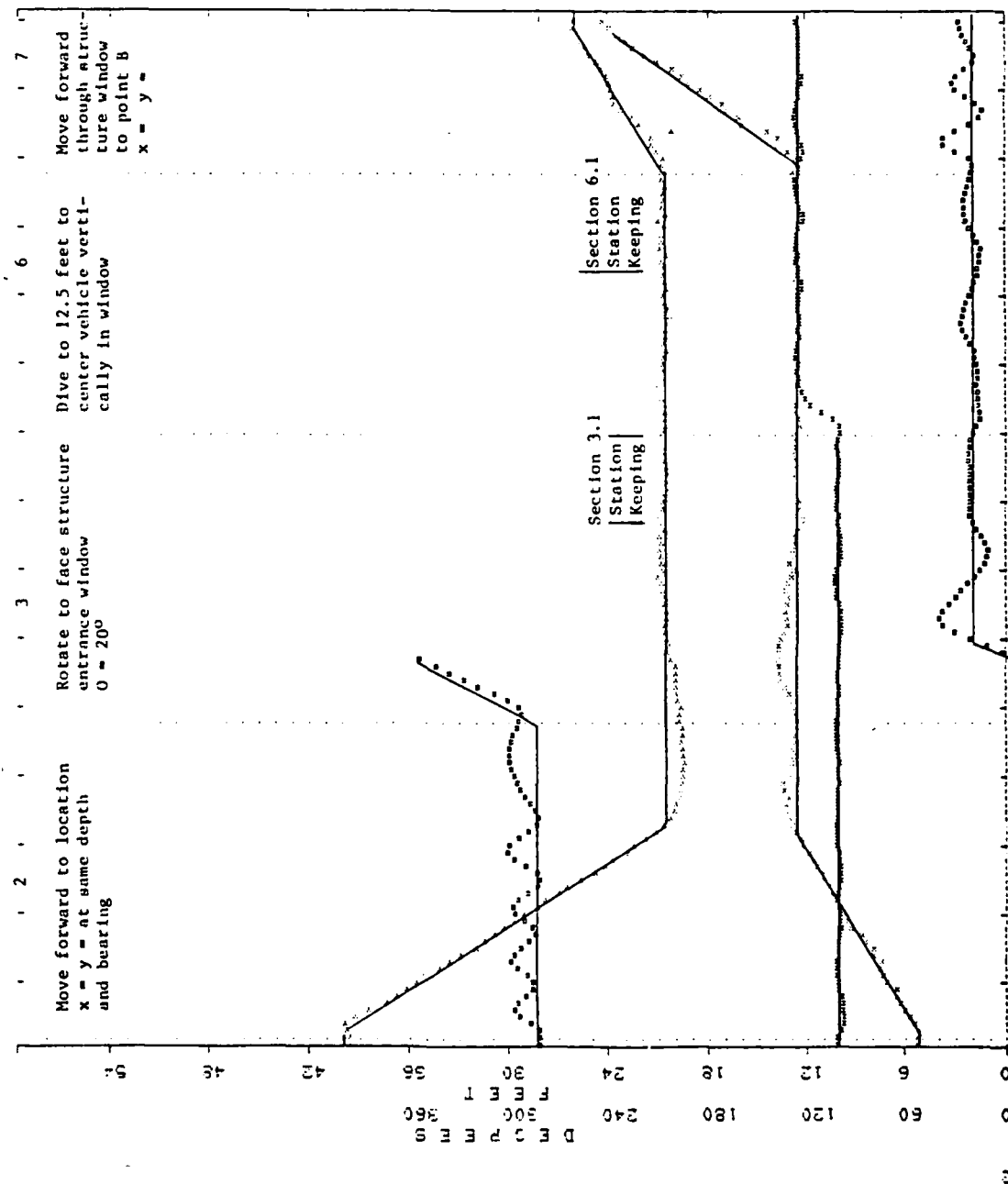
The system was partitioned into functional elements and wire-wrap boards were initially produced for CPU, memory and communication elements. These cards are universal elements that are part of any of the 68000 based systems, so printed circuit boards were then created for these functions (Figure 13). Additional wire-wrap cards were generated for the specific functions of interfacing to the compass and pressure transducer and the video digitizer and video display.

These cards have been integrated into two complete systems for the vehicle and one for video image processing. All three systems have been used almost continuously during 1982 and have proven to have excellent reliability.

6.2 Card Descriptions

6.2.1 68000 CPU (Figure 14 drawing #100187)

The 68000 CPU card contains the MC68000 microcomputer and interface buffers to the system bus. The full 68000 bus is implemented so that the CPU card can be used in a fully expanded system with other CPU's and DMA devices. A faster 68000 can be used with a higher frequency crystal. All logic, except the CPU and clock oscillator, has been implemented in high-speed CMOS to conserve battery power.

An auxiliary CPU support card connector has been provided on the top edge of the card. A ROM/RAM, UART support card can be used with the CPU for a two card system configuration. The local data bus at this connector allows the two card system to operate while debugging nonfunctional cards on the system bus.

COMMAND COMPUTER OVERVIEW

FIGURE 12

68000 Command Computer

68000 CPU Card
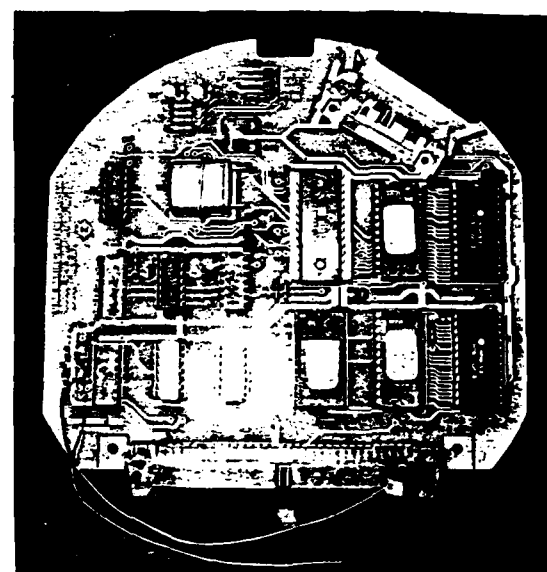
68000 CPU Support Card
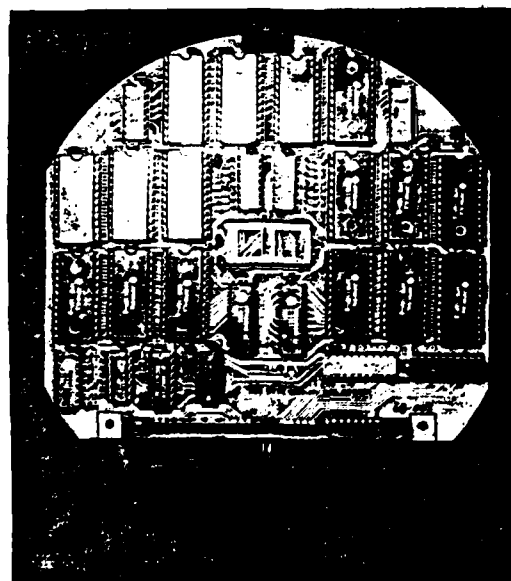
68000 ROM/RAM Card
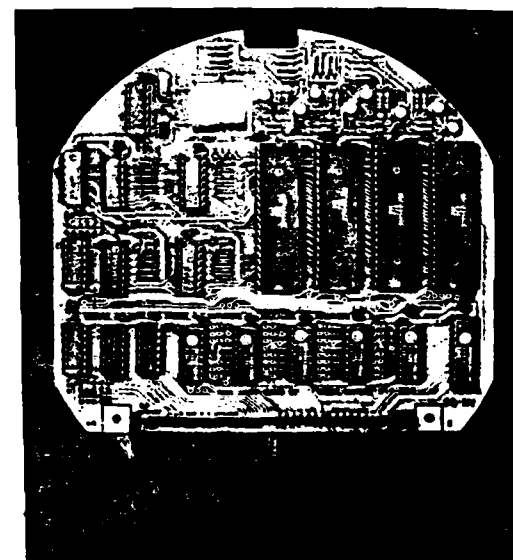
68000 UART Card

FIGURE 13:   68000 COMMAND COMPUTER

38

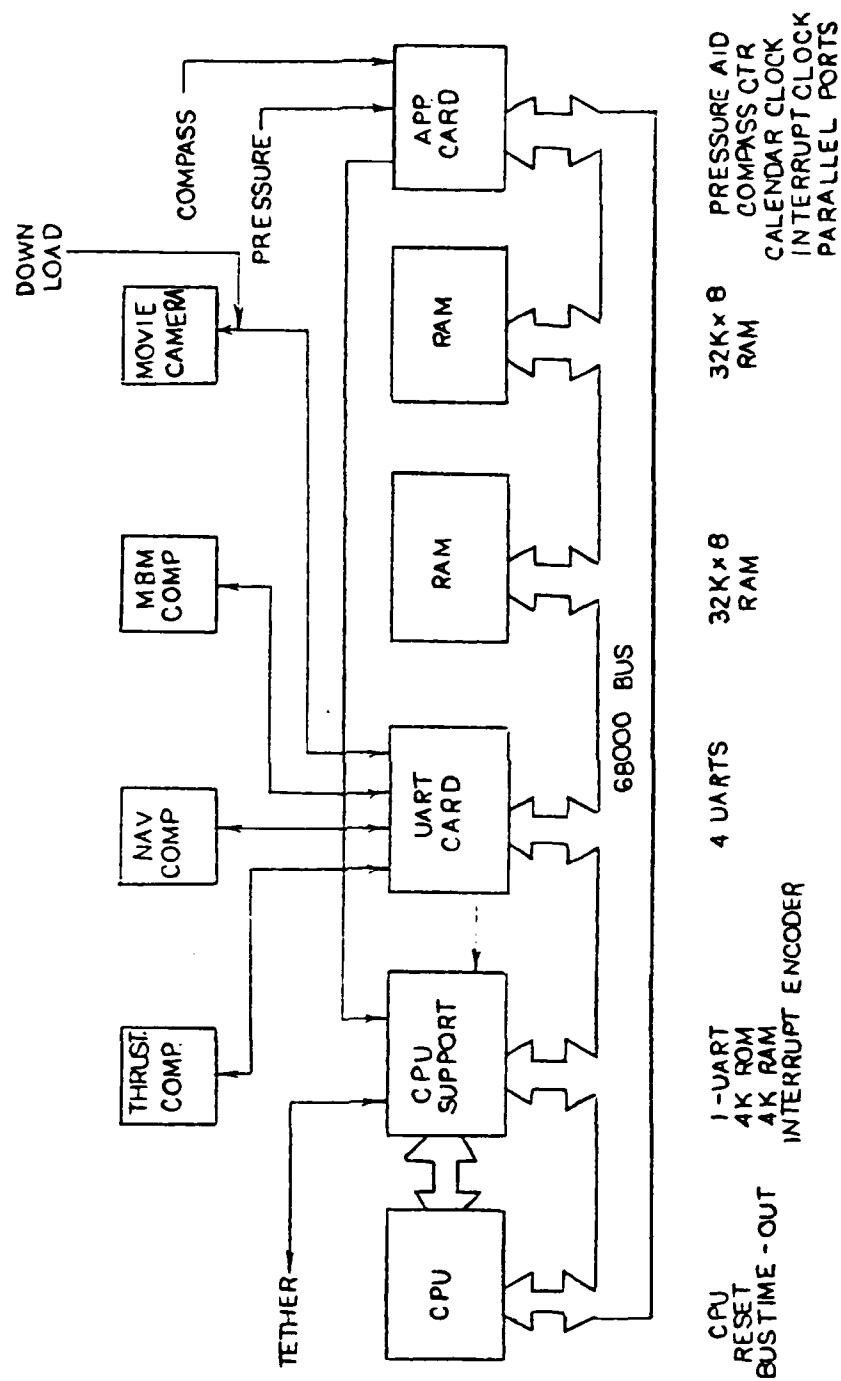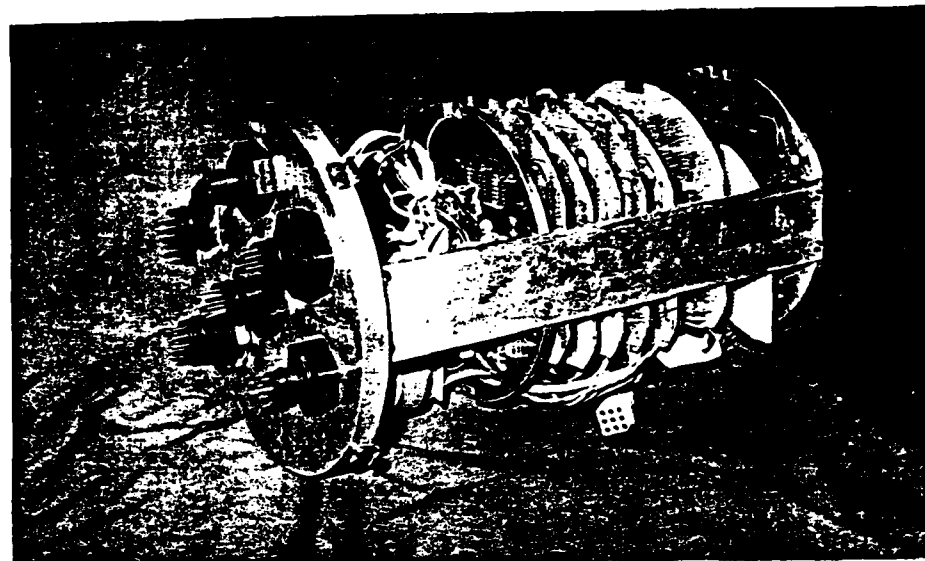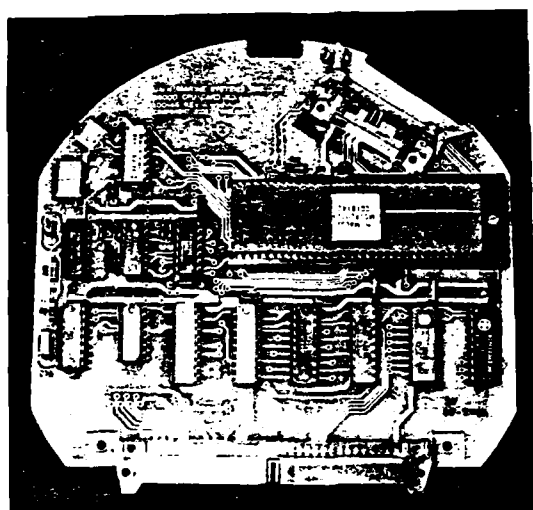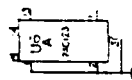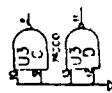The central component of this card is U5, the MC68000 CPU in its zero-insertion force socket. This receives a 4 MHz clock generated by crystal oscillator X1 and 74LS04 inverter U2. The CPU communicates to the bus via the address/control buffers U8 - U11 and bi-directional data buffers U12 - U13. These tri-state buffers are disabled when the bus grant signal (BG) is asserted by the CPU. Open collector 74C906 (U4) supplies the signal both to the bus and the tri-state controls so that either the CPU or another bus device can disable bus transfers. A pull-up resistor (part of DIP pack U1) holds this line normally high.

The unbuffered data bus is provided to J1 for auxiliary CPU support functions. Bus buffers U12 - U13 can also be disabled by pulling low the data bus enable (DBEN) line on J1 pin 3. This allows the support card to take control of the local data bus and disable the system data bus. The ENABLE for U12 - U13 is gated by 74C00 (U3) to combine the DBEN and BG signals. Resistor R2 keeps DBEN high when no support card functions are needed.

The CPU card also provides RESET and BUS ERROR logic. The reset can either be external or via the on-card push button reset. To reset the 68000 both HALT and RESET must be pulled low simultaneously. The open-collector buffer sections U4-D and E assert this low when either section A or C is pulled low. The A section is driven by a switch debounce flip-flop, U7-B. External reset is applied to U4C and the power up circuit formed by C6, R3 and D1. This circuit holds PBRESET low until power-up is complete. D1 discharges C6 when power is removed.

Bus error detection circuitry is provided by U6 and U7. The 68000 performs asynchronous bus cycles which wait for acknowledgment from memory or I/O devices. In the event that no device responds to a bus cycle and no DTACK or VPA signal appears, the CPU will wait forever and all operation will be suspended. This clearly is not desirable, so a bus time-out circuit is provided to monitor bus activity. As long as the address strobe signal (AS) is active, the one-shot (U6) keeps being re-triggered and its output flip-flop (U1A) is reset periodically. If AS does not return to a logical high for 1 ms or more, the one-shot completes its cycle. The Q (inv) output of the one-shot then goes high, clocking a high into the flip-flop output. If the CPU has not been halted, this input to U3A causes the BERR input to go low and generates a bus error trap in the 68000 CPU.

6.2.2 CPU Support Card (Figure 15 drawing #100184D)

The 68000 CPU support card (Figure 16) contains the ROM, RAM and I/O support needed to make the 68000 CPU card into a two-card core system. In addition, it contains an interrupt priority encoder which can be used by other cards in the system.

This card connects to the CPU card both through the system bus and through a local data bus at the top of the board. This local data bus allows the core CPU system to work even if a

40
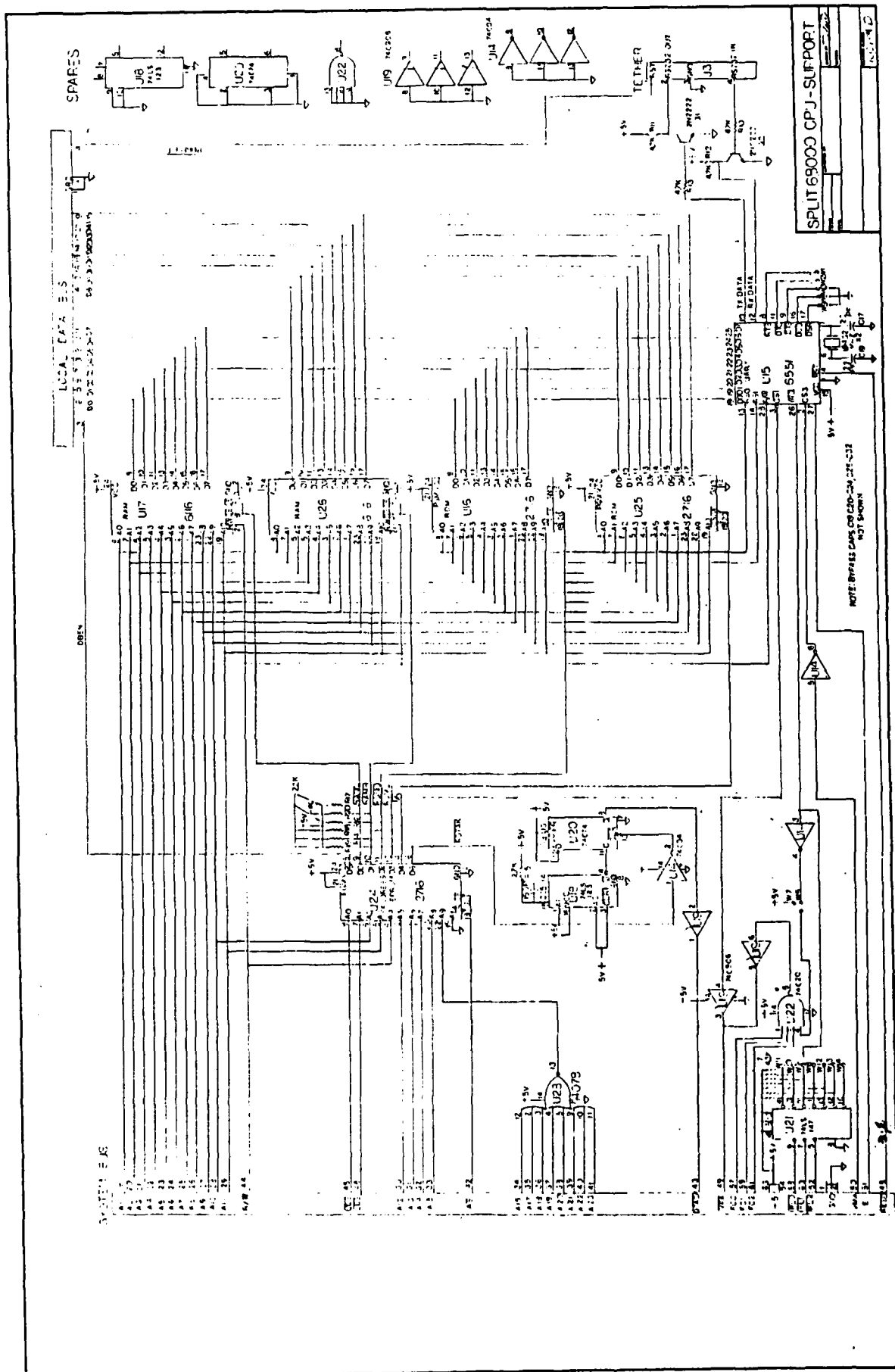
FIGURE 15 - 68000 CPU Support Card

41

68000 CPU SUPPORT CARD
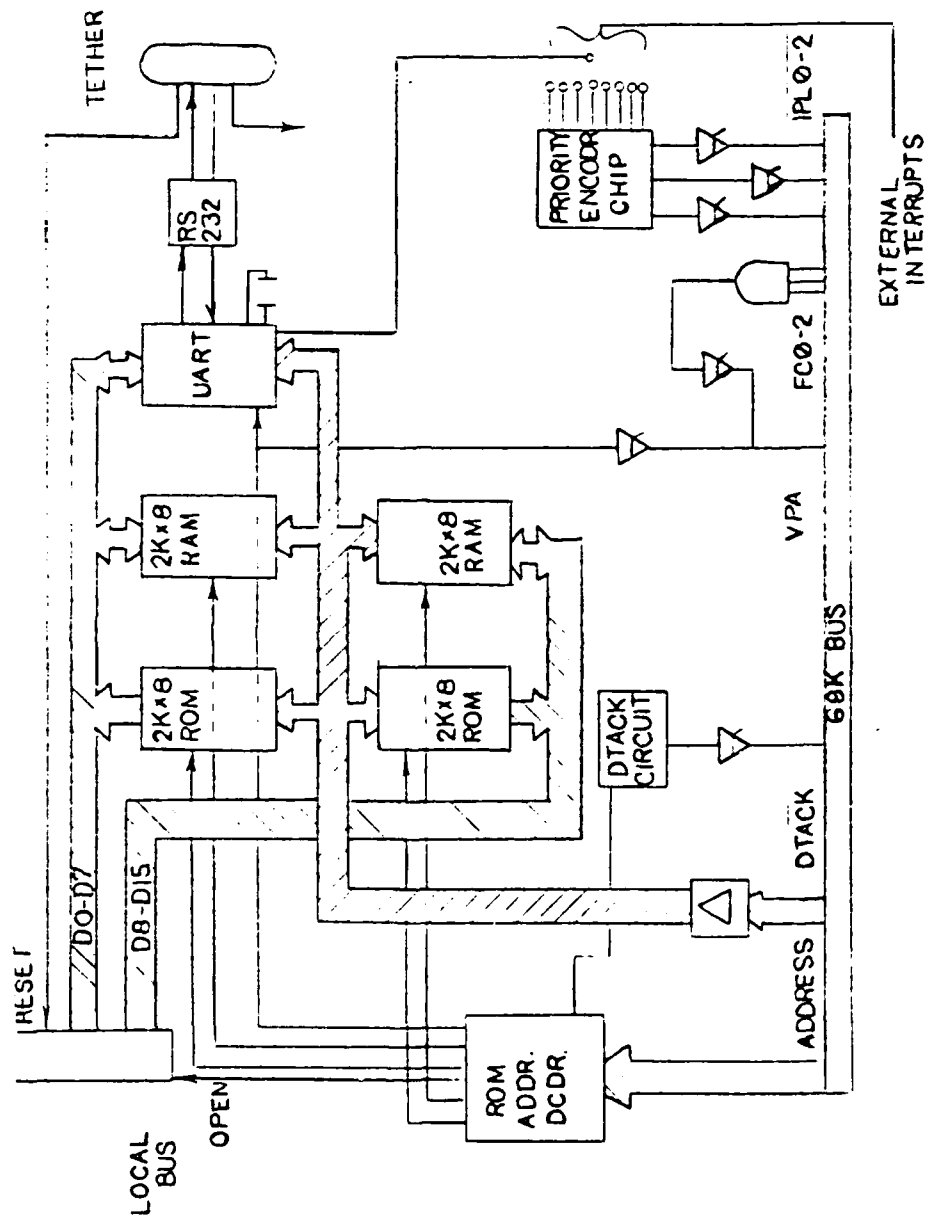
FIGURE 16

faulty card has disabled the data lines on the system bus. This feature is useful for bringing up new cards which may have problems.

The address space utilized by the on-card memory and I/O is determined by a programmable ROM. This allows much flexibility in address mapping. The EPROM at U24 supplies enable signals for the on-card RAM,ROM and UART and provides the local bus enable and acknowledge signals. It can be programmed to provide bus error detection if a write to ROM is attempted.

A system monitor resides in the EPROM's at U16 and U25. For simplicity, this monitor normally resides in the first 4K segment of address space, allowing the 68000 to execute monitor code immediately on power-up. All system traps are vectored into the support RAM at U17 and U26. ROM and RAM address decoding are provided by the address EPROM and are accompanied by an RDTAK output from the EPROM. This causes one-shot, U18, and flip-flop, U20 to generate a DTACK acknowledge signal which is supplied to the CPU by open-collector buffer U19.

Console I/O for the CPU system is provided by programmable UART, U15. This device contains a built in baud-rate generator with software selectable rate. A jumper block, W1-W5 allows access to handshake signals if they are needed by the host. Pseudo RS-232 levels (0-5V) are provided by transistors Q1 and Q2. These levels are compatible with most RS-232 devices and eliminate the need for non-5V supplies. Access to the UART is accompanied by assertion of VPA which causes the 68000 to execute a synchronous peripheral cycle which is required by the UART. A reset line is provided in the RS-232 cable which is passed through the auxiliary bus to the CPU. This allows the system to be reset over a remote tether.

Interrupts from the UART and from other cards in the system are processed by priority encoder, U21. This produces a coded interrupt level on IPL0 - IPL2 for the highest level for which there is an interrupt input. Jumpers W11 - W14 allow selection of the priority level for the UART. External interrupts can be wired onto the appropriate jumper post. U22 decodes the interrupt acknowledge state of the 68000 and asserts VPA through U19. This causes an interrupt auto-vector to occur in the 68000 which transfers program control to the vector for the acknowledged interrupt level. Jumper W6-7 allows selection of auto-vector for the UART only (W6), or for all interrupts (W7). If W6 is used, the other interrupt devices must assert their own acknowledge.

6.2.3 ROM/RAM CARD (Figure 17 drawing #100149)

The 68000 ROM/RAM card contains sockets for 32K Bytes of 2K x 8 memory devices. The memory is organized as 16K words of 16 bit data. Two alternative methods of address decoding may be selected by wire jumpers. The simplest and fastest method is to leave the socket for U15 empty and to install jumpers W25, W26,

43

FIGURE 17 – 68000 ROM/RAM

44
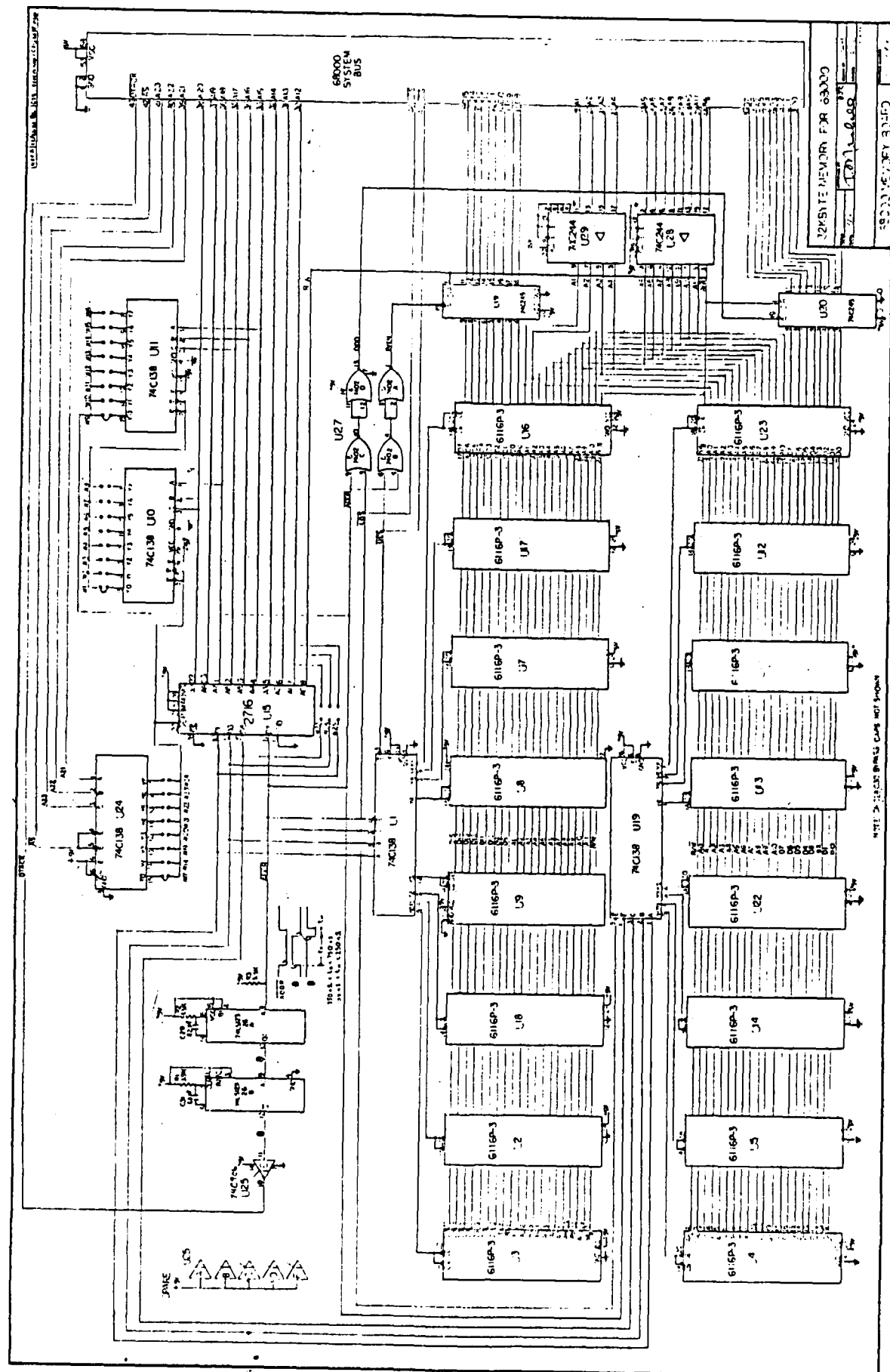
and W27. In this mode, the card can be addressed to any 32K Byte block on any 32K Byte boundary in the system address space. Address decoding begins with U24 which decodes bus address lines A21 - A23. The eight possible states of these three lines appear as outputs on jumpers W17 through W24 when Address Strobe (AS) is asserted. The desired jumper allows U10 to be enabled. This decodes A18 - A20 on wires W1 through W8 and enables the third decoder, U11. Jumpers W9 - W16 select the desired state of A15 - A17. The individual devices in this 32K block are selected by A12 - A14 which are connected through W25 - W27 directly to U1 and U19. These two remaining decoders gate upper data strobe (UDS) and lower data strobe (LDS) to the appropriate memory device. UDS and LDS are also gated by U27 with the address decoder output to provide enables for data bus buffers U19 and U20. Low order address lines to the memory devices are buffered by U28 and U29.

The decoded address also activates a dual one-shot, U26, which waits a delay period (determined by R2 and C29) and then returns a DTACK signal of width determined by R1 and C31. The first one-shot should be set to be slightly longer than the access time of the slowest memory device on the card. The second one-shot should be set to about one clock period of the 68000 CPU (250 ns for a 4MHz system).

The alternative address decoding method uses an EPROM to replace address decoders U10 and U11. Using this scheme, the highest address lines A21 - A23 are still determined by W17 - W24, but the 32K bytes of memory can be assigned in 4K blocks to any part of the 2 MByte address space. Any 4K block can also be designated as read-only. To use this address decoding method, remove U10 and U11 and jumpers W25 - W27 and place the address decoding EPROM in U15. This EPROM is coded with the block select in D1, D3 and D5 and the enable as a low in D7. The EPROM is addressed by R/W and A12 - A20. Thus the first two locations represent write and read of the first 4K segment, etc. Multiple mapping is allowed so that a block of memory may appear in two or more different segments. Memory can also be mapped to be read-only in one segment and write-only in a different segment. The only major drawbacks to this decoding scheme are slow-speed and the added power consumption of an NMOS or fusible link PROM. Fast CMOS EPROM's may reduce the consumption, but the chip is active at all times and with NMOS increases the board consumption from 30ma to 140ma.

6.2.4 UART CARD (Figure 18 drawing #100148)

Communication with other computer systems or peripheral devices is provided by the UART Card. Since power conservation is vital to many of the system applications, this card is implemented with CMOS UART's. The address of the four UART's is determined by U14 - U19 and jumpers W29 - W60. To simplify address decoding the 8-input NOR and U14 limits the address range to be under 020000 hex. Address lines A5 - A16 are directly decoded by U15 - U18 and any 32 address block within this space

45

FIGURE 18 - 68000 UART

can be selected with 4 jumper wires. The remaining decoder (U19) selects one of the four UART's. Each UART appears at two consecutive odd addresses and data is transferred over the lower data bus, DO - D7. Four programmable baud-rate generators, U3, U4, U10 and U11, provide the 16x clocks required by the UART's. Jumpers W1 - W8 and W15 - W22 allow a wide range of rates to be selected for each of the UART's. Transistors Q1 - Q8 provide a pseudo RS-232 interface that will accept RS-232 inputs and provide a 0 -5V output that is acceptable to most RS-232 devices. These interfaces allow the system to be used with a single 5V supply.

When a UART address is decoded, the VPA line is asserted by open-collector buffer U1-B. This causes the CPU to supply a synchronous bus cycle to the UART's and satisfies their internal timing requirements.

Interrupts from the four UART's appear at jumpers W9 - W14 and W23 - W28. A simple encoding scheme can be implemented with these jumpers, but multiple levels of interrupts can generate ambiguous codes in a system that does not employ priority encoding. For most implementations, the four interrupt outputs available at W9, W12, W23 and W26 should be used to drive a priority encoder such as the one available on the CPU Support Card. The open collector outputs allow more than one device to be wired to the same interrupt level input. An interrupt acknowledge (IACK) is generated on the card which asserts VPA when the CPU function code bits are all high and one of the UART's is generating an interrupt. This circuit may cause problems in a structured interrupt system since the processor may not be acknowledging the UART's interrupt if another device has higher priority. If all interrupts use the VPA auto-vectors, this is probably of no consequence, but if vectored interrupts are implemented or if the interrupt must be synchronized, the output on U1 pin 2 should be lifted by bending the pin up out of the socket. (This is pin 13 of a 74C906 if this part is used instead of the 7417).

6.2.5 Applications Card (Figure 19 drawing #100154)

The application card is a wire-wrap board which contains all of the hardware that is unique to the command computer of the SIMS vehicle. The card provides interfaces to the compass and pressure transducer and supplies both calendar and process interrupt clocks to the system.

Address decoding for the card is provided by U19 - U22 and is hard-wired to respond to 00C000 hex through 00C7FF hex. A DTACK signal is generated by U23 whenever this address block is decoded.

Additional address decoding by U19 and U20 generates enable signals for four 16-bit input and output ports at C080 through C087 hex. A 16-bit input port (U9-10) and output port (U11 - U12) is available to the user at C080-1 hex. Address C084-5 hex

47
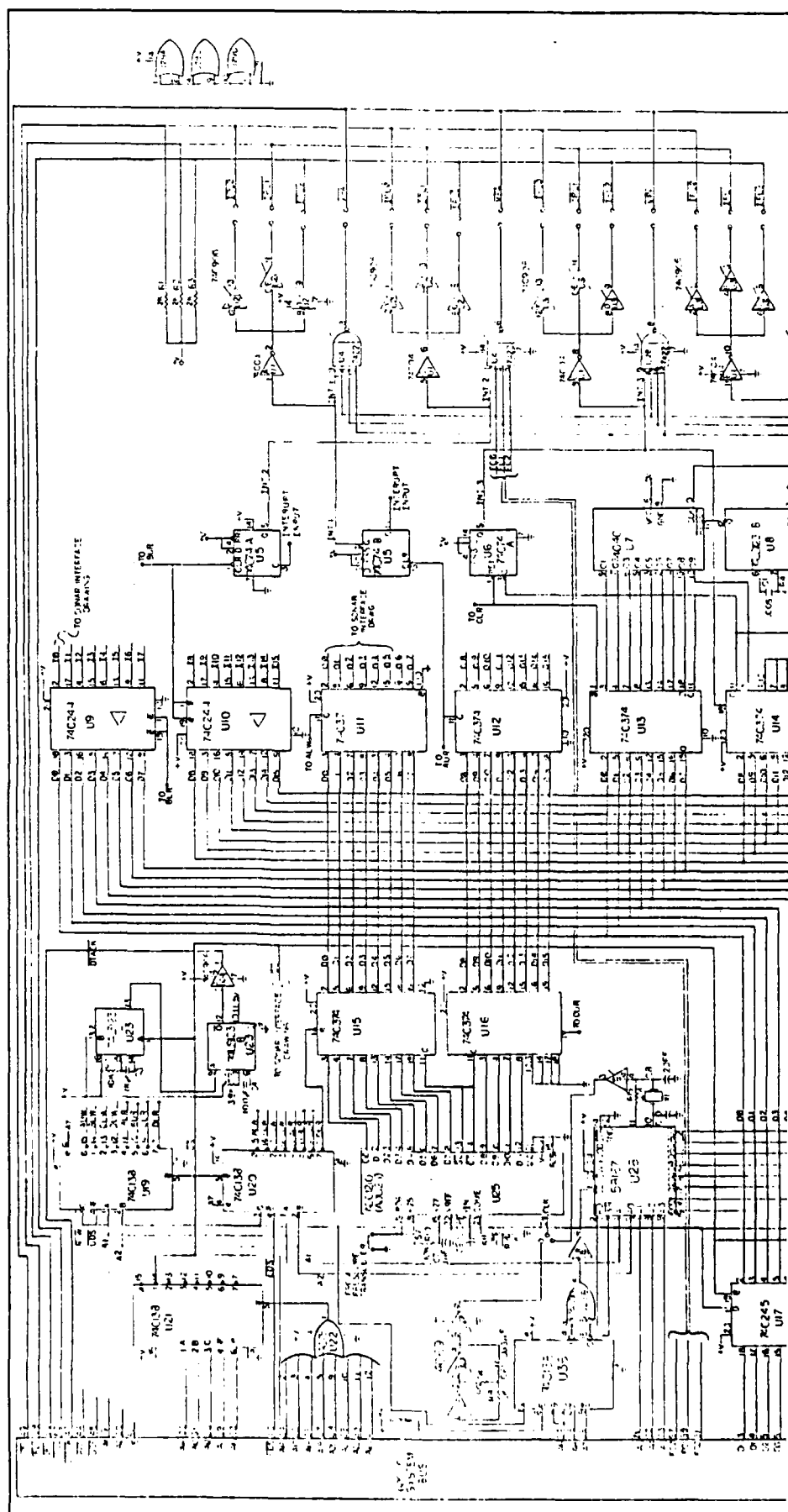
supplies the magnetic compass bearing from compass counter circuit U7 - 8 and latches U13 - 14 (Figure 20). The analog depth signal from the pressure interface card is digitized by 12 bit ADC U-25. The digital output is latched by U15 - 16 and made available at C086-7 hex. The code is inverted by the A/D converter and must be complemented by the user. Address decoder U36 selects the calendar clock (MM58167), U26, at address C000 - C07F. The 32 registers in this clock chip can be read or written to at odd addresses from C001 to C03F. This clock maintains the date and time to milliseconds and can provide interrupts on a preset date-time or at each of the unit intervals (once per second, minute, hour, day, week or month). The clock is used for logging data to the bubble memory but can also be used to provide sleep functions or mission abort at some cut-off time.

An additional timer/counter is provided by an RCA 1878 which appears at odd addresses C101 - C10F. This dual timer can be programmed to count a 32 kHz clock to provide real-time interrupt to operating system software for multi-task switching. The second section of the timer is available to be used for external applications such as an echo return timer for an auxiliary SONAR interface.

6.2.6 <u>Pressure</u> <u>Interface</u> (Figure 21)

The command computer measures the vehicles depth in the water by a pressure transducer mounted on the bulkhead and vented to outside water. A Setra systems #104 strain-gage transducer provides a differential analog signal to the pressure preamplifier mounted on the rear of the transducer housing. This preamplifier converts the differential signal into a 0 - 5V signal acceptable to the 12-bit A/D converter on the application card. A precision reference provides adjustable offset.

During initial testing, the pressure interface displayed severe nonlinearity and poor repeatability in measurement. This was traced to digital noise in the A/D converter from its being wire-wrapped and in close proximity to other logic. The successive approximation converter used in this design is prone to such errors. A reasonable solution was achieved by moving the A/D converter to a separate wire wrap board behind the application card and keeping all analog signals away from the application card. This improved accuracy to a consistent 10 bits with two LSB's uncertain. To obtain the full range of the A/D converter input, a negative supply was needed for the A/D's internal comparator. This was supplied by a charge pump circuit running off the system clock. The final result was a system with 10 bit linearity as shown in Figure 22. The accuracy was calibrated at room temperature and normal voltages and found to be repeatable. An accuracy of 1/2 inch in 60 feet was measured.

6.2.7 <u>Pressure</u> <u>Transducer</u> <u>Test</u> <u>Results</u>

After field tests were terminated in December 1982, a laboratory test was performed to determine linearity of the

COMPASS INTERFACE

FIGURE 20

COMPASS INTERFACE
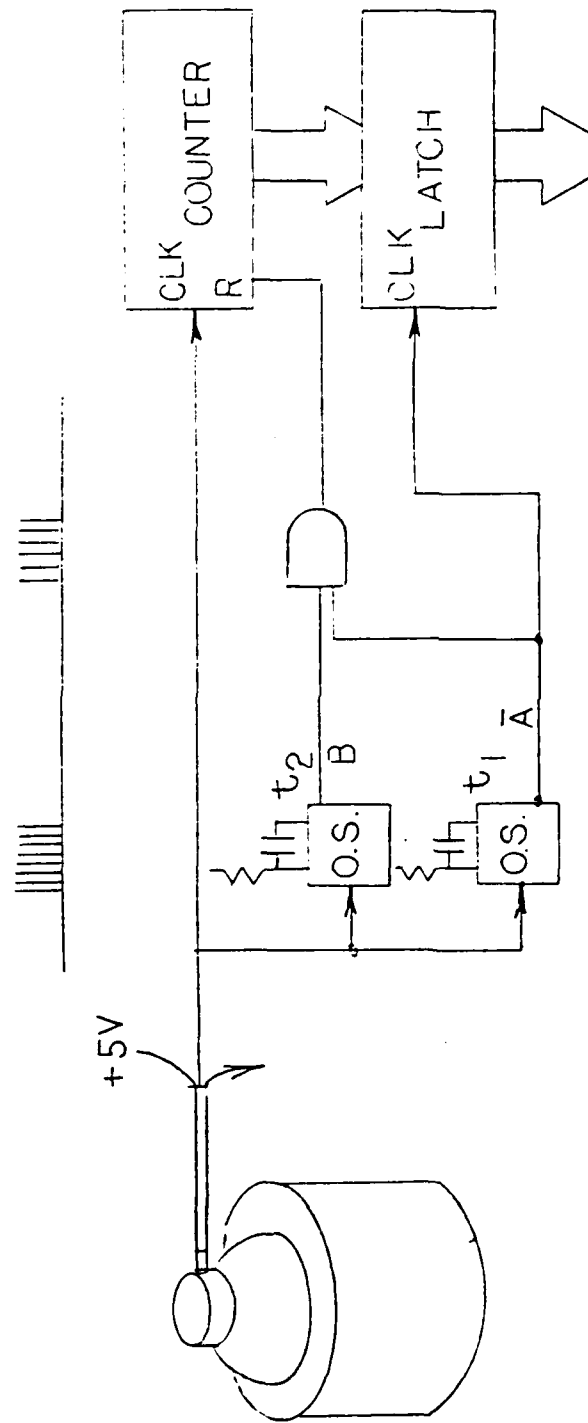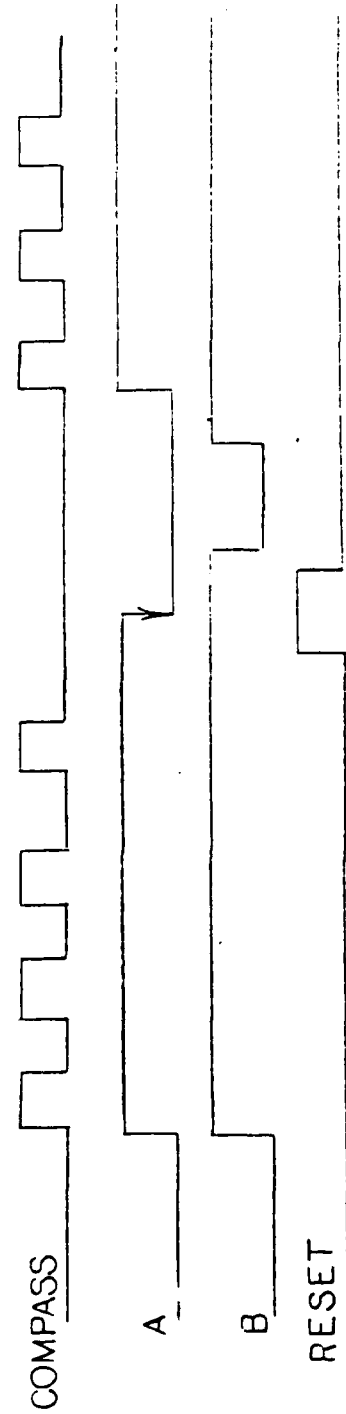
1-360 PULSES
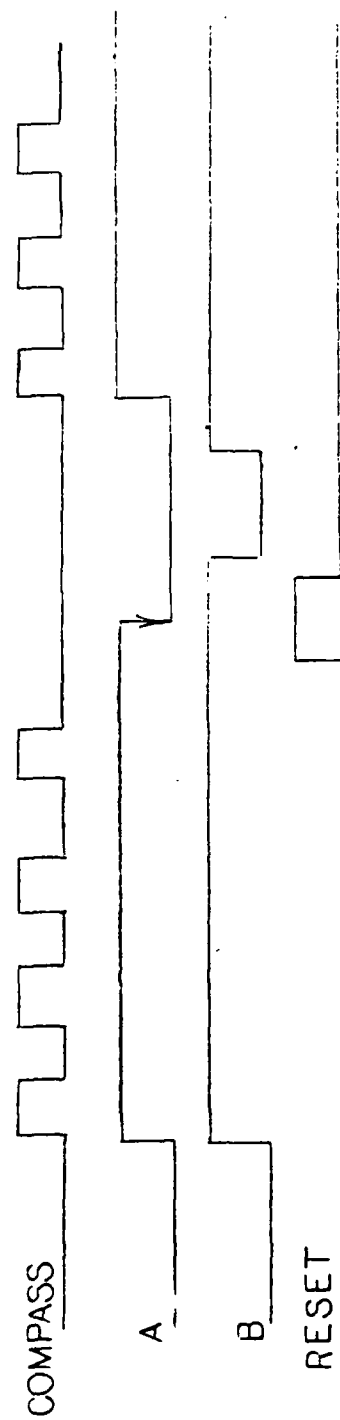
FIGURE 20

PRESSURE TRANSDUCER INTERFACE
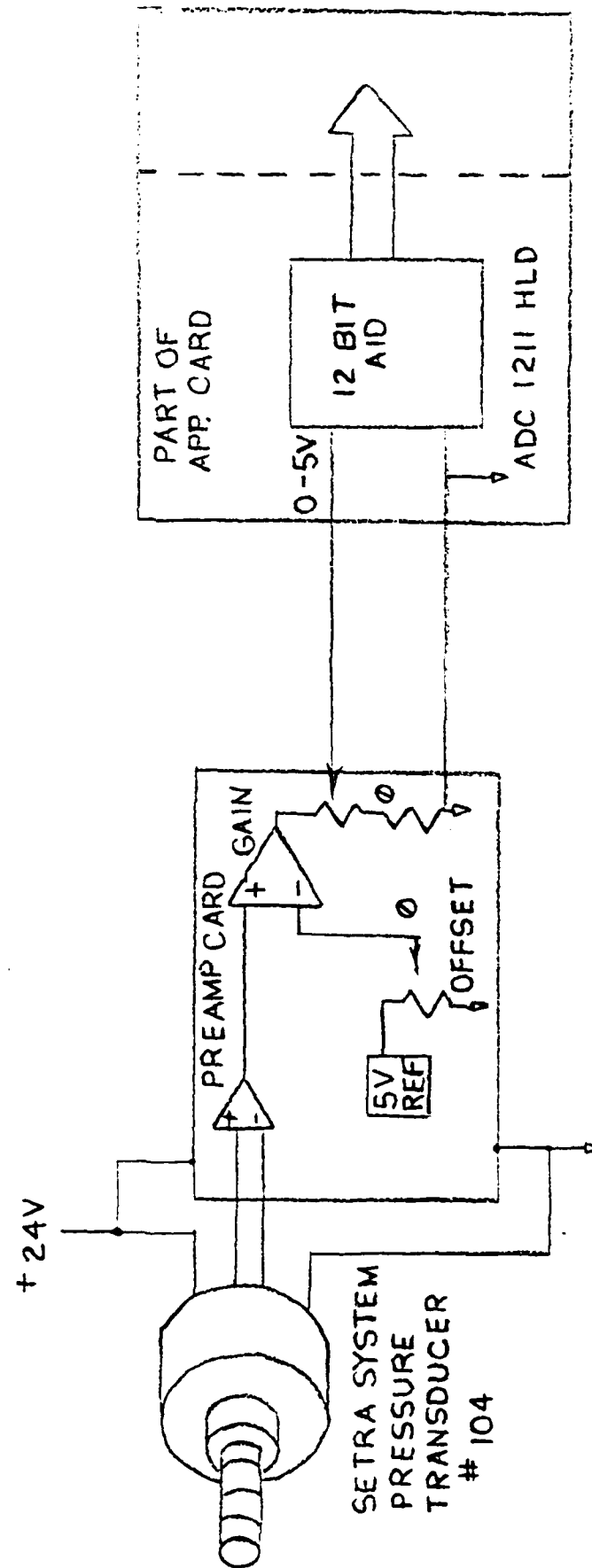
FIGURE 21

51

DEPTH MEASUREMENT LINEARITY
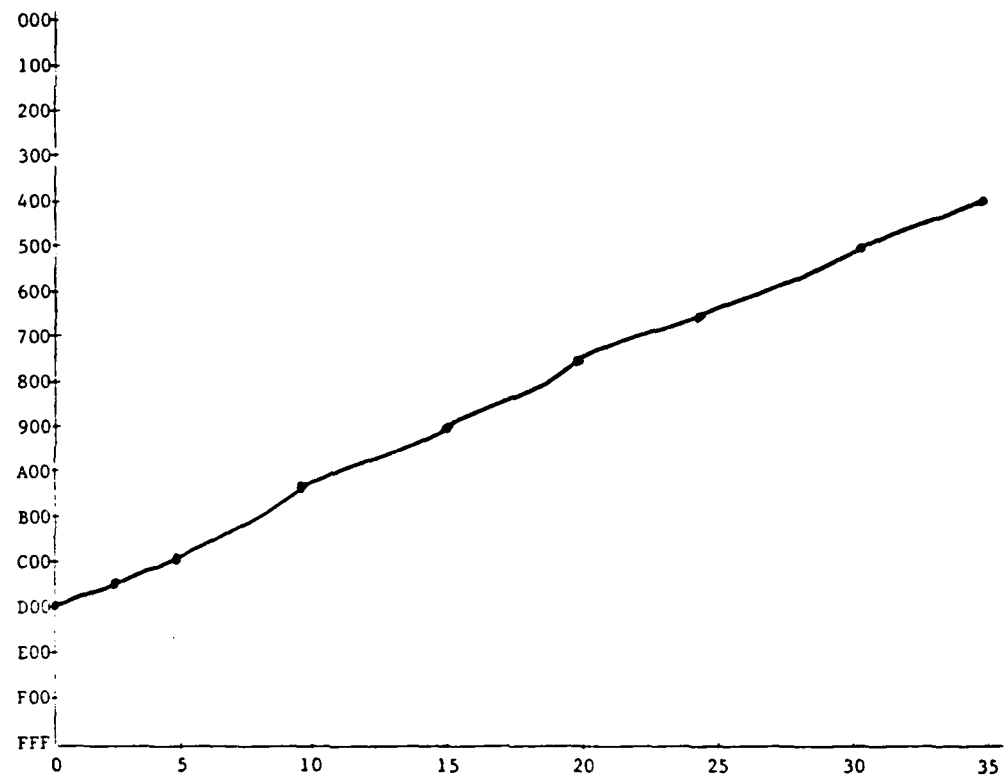
FIGURE 22

52

vehicle depth sensor system (pressure transducer/A/D converter) as well as its sensitivity to voltage and temperature changes.

It was immediately found that a built in error of as much as 16 counts was being introduced by an incorrect decoupling resistor (100 ohms instead of 10 ohms) in the ADC ±5 volt supply line.

The test data shows that for a system voltage range of 20-24 volts and a temperature range of 40 degrees F - 68 degrees F the depth sensor system is accurate to +3.3 inches and -1 inch.

6.3   Image Processing Systems (Figure 23)

An important part of the SIMS system is the desirability to send images to the surface. The limited bandwidth of the acoustic channel available for this purpose will require significant bandwidth compression. Exploration of image processing techniques has been investigated using our vehicle CPU system with a 100 x 100 CCD camera and a video digitizing board and video display board. These wire-wrap cards are currently being used in one system to investigate compression techniques. The digitizer card could be placed on the vehicle in a dedicated image transmission system and the display board installed in a second 68000 system on the surface. A more detailed description of the imaging system is described in Section 16.

6.3.1   Video Digitizer Board (Figure 24 drawing #100181)

This card receives video and timing information from a 100 x 100 CCD camera and digitizes the image into 10,000 six-bit pixels. These pixels are stored in 16K of on-board CMOS RAM, U1 - U8.

The video, pixel clock, horizontal and vertical sync signals are supplied to the card through a connector for camera input. The pixel clock is gated with HSYNC and supplied to pixel counters U20 - U21. These counters are reset at each VSYNC. The video is supplied to a CA3300 six-bit CMOS flash converter (U23) and the digital data latched into U17. When a frame is desired, the frame grab flip-flops at U25 are activated. By accessing memory location 030000 hex, the CPU causes U25A to be set. On the next vertical sync pulse, this causes U25B to be set and causes the storage of digitized data to begin. U17 - U19 are enabled presenting pixel address and data to the RAMs (U1 - U8) which are placed in write mode. U25A is reset by U25B and at the end of the frame VSYNC again clocks U25B terminating storage. Access to the RAM at odd addresses 020001 - 027FFF hex is provided by U11 - U13 when this address block is decoded by U14 - U16. Access during a frame digitization will reset U25B and cause immediate termination of the grab. This leaves the memory only partly filled and should be avoided. Status information is provided at address 028001 hex by U10 to allow video software to know when a valid frame has been stored.

C.C.D. VIDEO SYSTEM

FIGURE 23

54
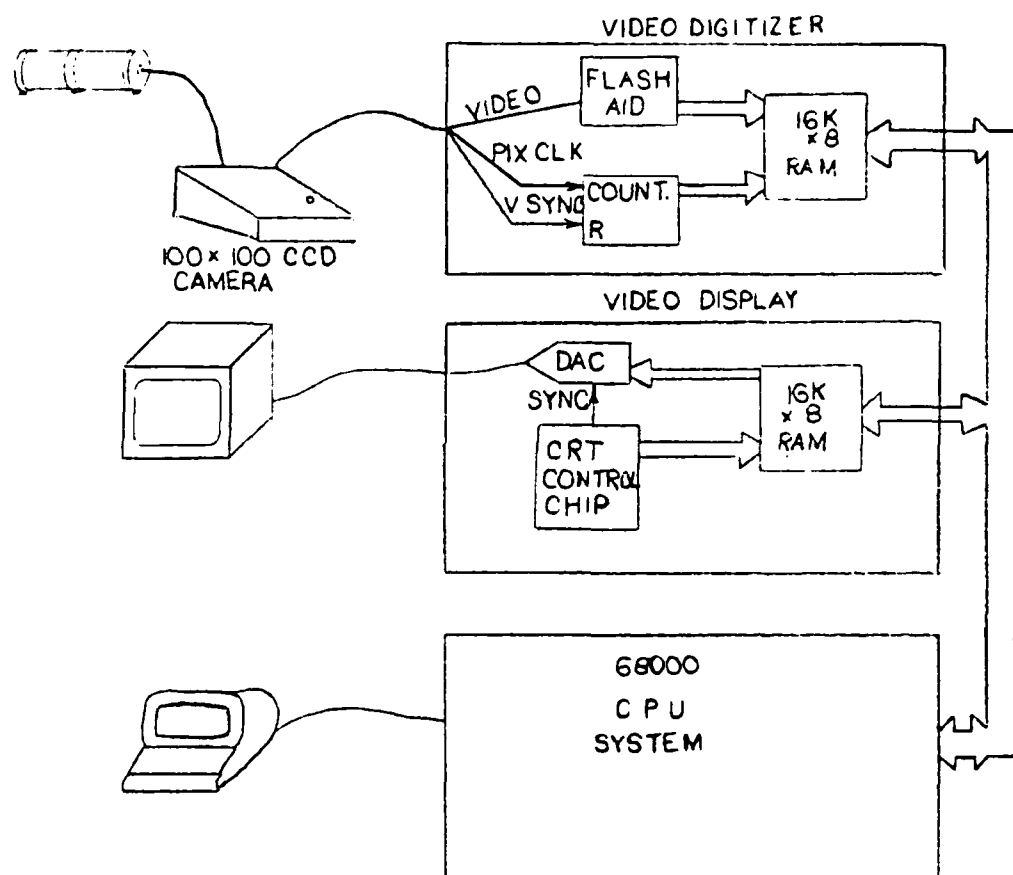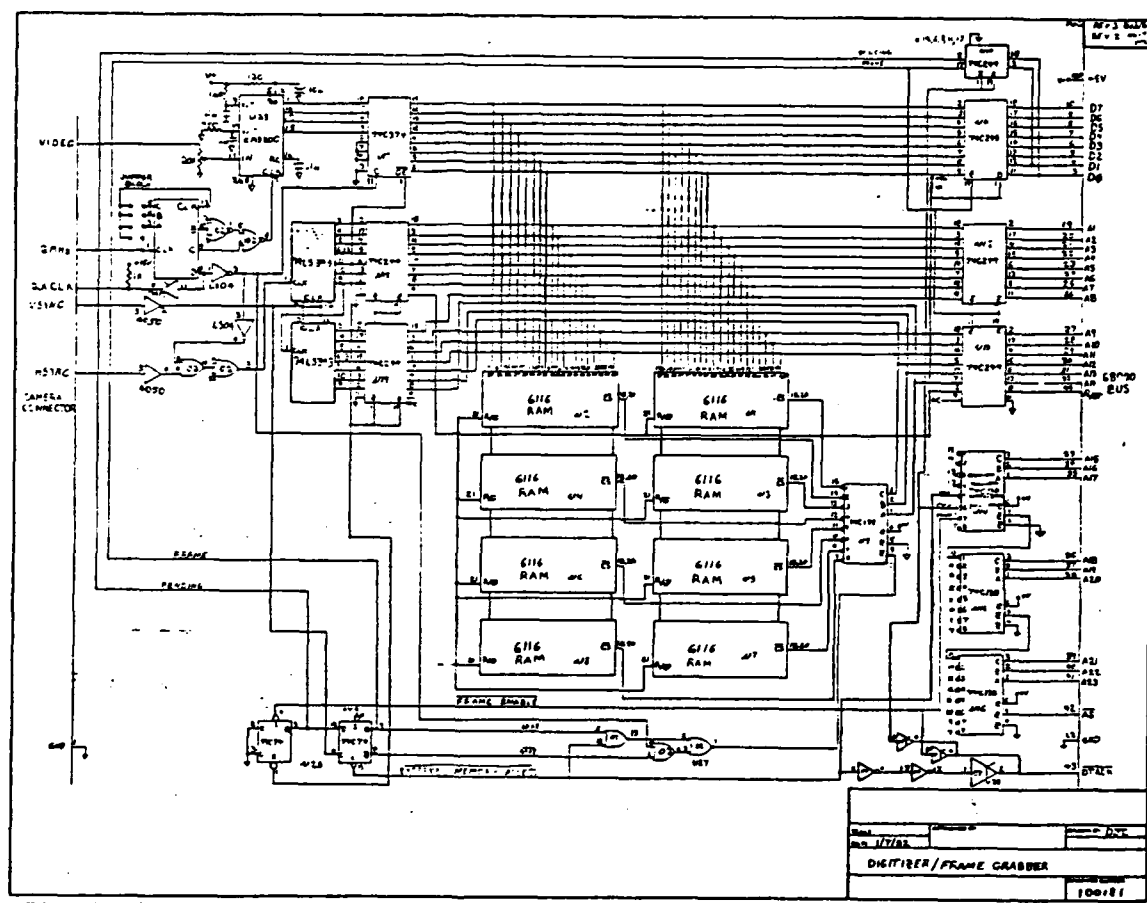
FIGURE 24 - Video Digitizer

6.3.2 Video Display Card (Figure 25 drawing #100177)

The digitized video from the 100 x 100 CCD camera is processed and then displayed. To be compatible with standard video equipment, the display must be a standard NTSC video output. A Motorola 6845 CRT controller, U3, is programmed by the CPU to generate the proper timing for a 100 x 100 video display. The 6845 registers are accessed at address 01F001 and 01F003 hex by decoders U13 - U14 and bus buffer U30. The CRTC supplies pixel addresses to the 16K on-board RAMs U19 - U26 through address buffers U11 and U12. The data for each pixel is latched into U28 and formed into video by an 8-bit resistor DAC. Sync and blanking are supplied by U5, U6 and Q1 and Q2 with diode D1 holding the video at the blanking level. The output is finally buffered by Q3. A strobe feature is provided by flip-flop U2 and latch U29. By writing first 00 and then 06 to address 01F2001 hex the CPU can cause U2 to gate only a single video frame to the display. This is very useful for photographing the CRT without bars in the picture. Writing 02 to this location enables the display continuously.

Transparent access to the display memory is provided between display pixel cycles. Access is synchronized by flip-flop U31. Whenever addresses 010000 - 017FFF hex are decoded by U15 - U17, the access sequence is started at U31. At the next inter-pixel interval U31A is clocked and the address buffers U9 and U10 are enabled. If the cycle is a write, the R/W line is asserted on the RAM's and data is supplied to them by U7. If the cycle is read, the data is presented to U8 and latched at the end of the cycle. The cycle ends when the next pixel clock cycle begins. The low at U31A is clocked into U31B and access to the RAM is switched back to the CRTC. At this time a DTACK signal is supplied to the CPU. For read cycles the CPU can now examine the data in latch U8. Write cycles are completed already. When the CPU acknowledges DTACK by removing the address strobe, U31 then returns to its original state and DTACK is removed.

6.4 68000 ROM Monitor

The program that runs when power is first applied to a microcomputer plays an important role in the system development. This program resides in ROM and is usually referred to as a "monitor". Once a system is fully operational, the monitor will frequently be used only to load and start more sophisticated software. During initial design or when debugging difficult hardware or software problems, the monitor should provide the tools needed to develop and maintain the system.

The first version of the 68000 monitor (developed in June 1981) provided the basic tools of a hex loader, memory examine/change, dump, trace and breakpoint. The next revisions included transparent communications with each of the UARTS, loading from port #3 and a random pattern memory test.

56

FIGURE 25 - Video Display

Revision 1.2 included buffered interrupt I/O on all ports, but this feature was abandoned in Revision 1.3 and higher since it consumed a fairly large block of space, and the same capability was being provided by the vehicle operating system which had been developed.

Initial versions through 1.3 were based on 65K of user accessible RAM. By July 1982 it became obvious that this was going to limit software development. The address input routine was then expanded in a new version (Revision 2.0), and a binary loader added to directly dump files from the ERG development system. The binary loader is three times faster than the original hex loader, can load the full address range, and can be called from within the "C" programs that have been loaded previously. It returns the next available location to the "C" program.

This monitor is the current revision (Figure 26). A modified version has been provided for the CCD camera system, which includes automatic set-up and operation of the CCD video digitizer and display boards. This is a separate mini-monitor embedded in some free space within the Revision 2.0 version.

Copies of user's manuals for Revisions 1.2, 2.0 and the video monitor appear in Appendix B.

6.5 System Design Considerations

6.5.1 Present Design Problems

The 68000 CPU system was designed with the concept of complete flexibility in mind. The CPU support card with its higher-powered NMOS circuitry was to be used as a starter system and removed from final low-power systems. Unfortunately, the interrupt encoder was placed on the support card. The card can be unstuffed except for U19, U21 and U22 and it will draw minimum power but unless the equivalent circuitry is provided elsewhere, the card must still be present on the bus to allow a prioritized interrupt structure.

The ROM/RAM cards were supposed to allow any 2716/6116 compatible part to be plugged into any location. The PD/PGM pin (pin 21) of NMOS EPROM's loads the buss down and prevents R/W operation. Since there is no room for jumpers on the PC layout, devices cannot be mixed on the card unless pin 21 of the EPROM's is bent out and wired to pin 24. This probably is not a problem with CMOS EPROMS. Since the pins on some EPROMS are fragile, an alternative method is to dedicate an entire card to EPROM's.

The ROM address decoder on the ROM/RAM card is not used because of the power consumption of sufficiently fast ROM. It could be used when really fast CMOS ROM's become available, but is bypassed in present implementations.

58

SUMMARY OF MONITOR COMMANDS

```
B aaaaaa              -   Set breakpoint at "aaaaaa" (RAM or ROM)
B K                   -   Clears all breakpoints

C                     -   Continue after a breakpoint

D aaaaaa bbbbbb       -   Display a block of memory

G aaaaaa              -   Go jump to code at aaaaaa

HB                    -   Binary load tape on port #3
H L                   -   Load a HEX tape on port #3 (see L)
H P aaaaaa bbbbbb     -   Write HEX dump of aaaaaa-bbbbbb (see P)

I                     -   Enable interrupts

L                     -   Load a Motorola "S1" format HEX tape -
                          lower 65K only

M aaaaaa              -   Examine/change memory at aaaaaa

O                     -   Enter transparent mode with UARTS 0-3

P aaaaaa bbbbbb       -   Write Motorola "S1" tape of aaaaaa-bbbbbb

R                     -   Display DO-D7 and AO-A7 from last break

S aaaaaa bbbbbb dd -      Set "dd" into block of memory aaaaaa-bbbbbb

T aaaaaa bbbbbb tt -      Set TRACE if PC is in aaaaaa-bbbbbb block
U                     -   Turn off TRACE

X aaaaaa bbbbbb
  cccccc              -   Block move aaaaaa-bbbbbb to cccccc-

? aaaaaa bbbbbb       -   Random pattern memory test of aaaaaa-bbbbbb
```

```
(ctrl C)   Restarts monitor
(ctrl S)   Stops output
(ctrl Q)   Re-starts output
```

FIGURE 26

The ROM/RAM card DTACK circuit works, but is of a poor design. The actual DTACK signal is of fixed duration, controlled by the second one-shot. A proper design would terminate DTACK when the CPU recognized it and removed address strobe (AS). The present design must be adjusted if a faster CPU is incorporated.

A system design problem exists in the use of the CPU support card with a ROM/RAM card addressed in the same space. To provide RAM at $2000 - $7FFF with the present ROM/RAM configuration, this card must respond to $0000 - $1FFF as well. Since the CPU support card has been configured to provide ROM and RAM in this space, an address decoding conflict exists. The support card uses a local data bus and disables the system bus so that no bus conflicts occur, but both CPU support and ROM/RAM card provide DTACK signals. It is therefore important to set the DTACK response time slightly slower on the ROM/RAM than on the support card. This slows overall system performance. This is eliminated if the support card is de-populated (as previously described), but the monitor ROM must then share the ROM/RAM card with RAM, and pin 21 of the EPROM's must be separated from the bus and held high.

The power consumption for individual cards is shown in Figure 27. Also included is a cumulative combined total power usage for the system.

6.5.2 Future Expansion

The bus architecture chosen for the SIMS 68000 computers contains all 68000 function pins. Any 68000 compatible device can be attached to this bus. This opens the door to future expansion with memory management devices, floating point co-processors and multi-port shared memory. Thus these same computer components can be used to build very powerful systems as the need arises for greater on-board intelligence. Future possibilities include image directed guidance input.

60

POWER CONSUMPTION

CARD	INDIVIDUAL	COMBINED*	WA INDIVIDUA
CPU	176 mA	---	0.88 W
SUPPORT	137 mA	348 mA	0.68 W
32K RAM	25.8 mA	358 mA	0.13 W
UART	19.0 mA	374 mA	0.09 W
APP CARD	217 mA	618 mA	1.09 W

*

 Combined systems are running and accessing the added
 Power consumption relfects operation.

FIGURE 27

7. 68000 COMMAND COMPUTER SOFTWARE

7.1 Overview

At the heart of the command CPU software is the Vehicle Operating System (VOS) shown in Figure 28.   The VOS performs two basic functions.   First, it handles all I/O system calls and, secondly, it controls the scheduling of various tasks running in parallel in its environment.   Always running in the O/S environment as a task is the system monitor.   The monitor receives and sends information across the tether to the operator TTY.   It also contains a binary loader which will download user software from the ERG development station.   Running in parallel with the monitor is the mission software package (MSP).   The MSP consists of a number of software modules that control all hardware system and run a user defined mission scenario. Surrounding the MSP is a shell of interactive software that acts as a user interface between the operator and the MSP.

All communication between the command CPU and the three peripheral CPU's is handled by interrupt driven system calls to the VOS.

7.2 Vehicle Operating System - An Internal Overview

The operating system (Figure 28) should first be isolated from the three other computers and the monitor and treated as simply switching a group of tasks and handling O.S. requests. After this is done, conceptually, the task of explanation is greatly simplified.   In the most simple case a group of tasks is running, each one after the other making no requests of the O.S. In this situation the O.S. starts a task by loading the countdown clock with that tasks time slice and then execution transfers to the task itself, restoring its registers in the process. Execution continues within that task until the countdown clock reaches zero and an interrupt occurs.   The interrupt service routine saves all that tasks registers in its processed descriptor block (PDB).   The service routine then calls a function which returns the PDB of the next task to run, this is normally the next task in the linked list of PDBs.   At present each task runs until it has completed execution, meaning there are no time limits associated with each task and they are allowed to run "forever".   When a task does come to the end of its execution its PDB is marked as finished and it is run no more. The situation becomes somewhat more complicated when the tasks running start making I/O requests and the like.   Everything proceeds as normal until an unsatisfiable request is made.   An unsatisfiable request is, for example, attempting a read of ten characters when only five are available.   When this happens the requesting task is put into a wait state, input wait in the case of a read.   Within the O.S.   there is a separate queue for each input and output channel, when a task goes into I/O wait it is placed in the appropriate queue.   The mechanism for making requests of the O.S. is fairly simple.   For example, a read

FIGURE 28

63
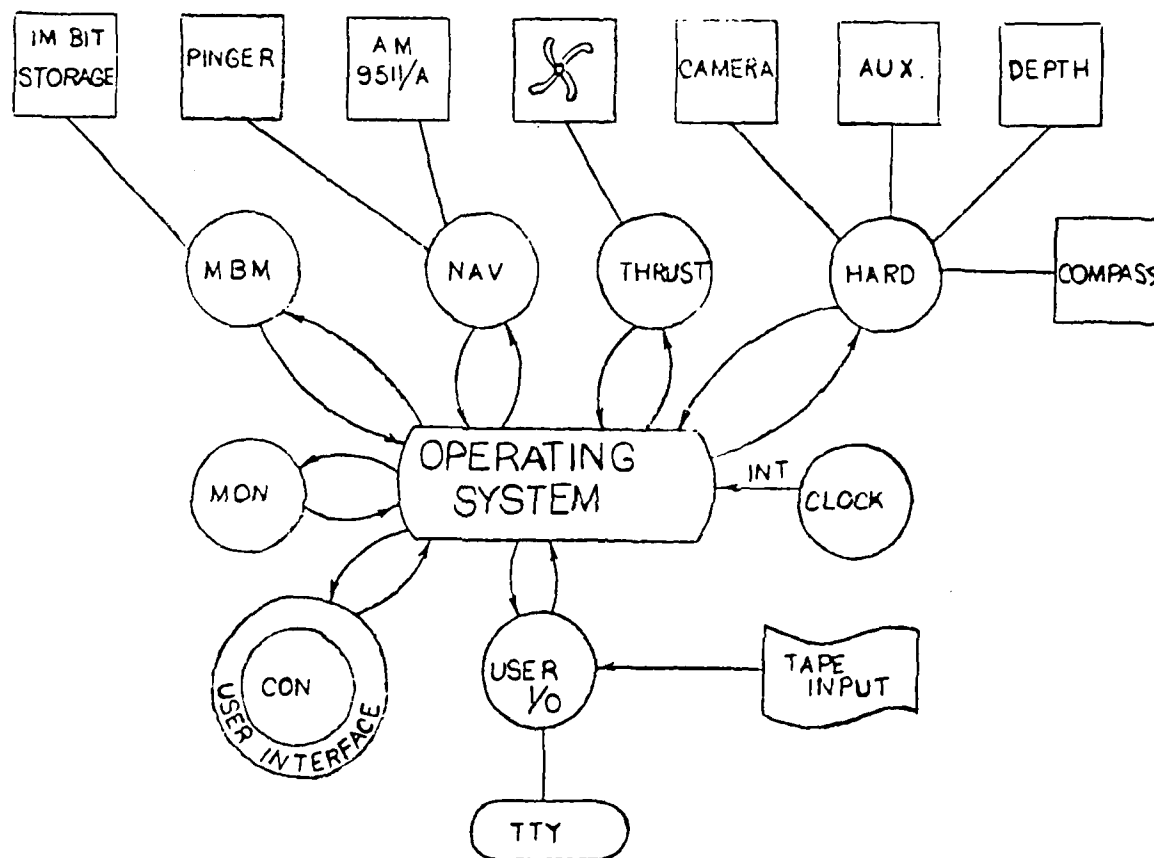
request is made via a subroutine call, e.g. read (fd, buff, nchars). At link time a routine, labeled read, is linked to the subroutine call. Where both C and Pascal push their parameters on the stack, on entry to the routine (_read), the return pc is on the top of the stack followed by the parameters for the call. The _read routine then saves the return pc in AO loads D7 with the index of the O.S. routine, three in the case of a read, and executes a trip #1. When the trap is executed the 68000 enters supervisor mode and the trap handling routine is entered. On entry to that routine the address of the appropriate handling routine is looked up in a table using D7 as an index. The trap handling routine sets up the stack with the two parameters for the individual handling routines. The first is a flag which indicates whether this is the first attempt at satisfying the request and the second is a pointer to the callers parameter block.

Note: Because parameters are always pushed on the stack as long words the parameter blocks defined in the O.S. should only contain structures made up of long words.

7.3 Vehicle Operating System Description

The Vehicle Operating System (VOS) is a single user operating system designed to be used in the UNH Marine Systems Engineering Lab's autonomous vehicle EAVE. The primary goal of the VOS is to allow the users of the vehicle to program in a high level language such as C, PASCAL or FORTRAN. VOS is structured in such a way as to minimize or eliminate the hardware knowledge required to utilize the full capabilities of the vehicle. The operating system is currently configured to run with the following hardware:

a 6551 - console uart

3 - 1854 - communication uarts connected to three 6100's which control the thrusters, provide navigation information, and record data on the MBM.

a countdown clock - used to do task switching

a datetime clock - used to keep track of the time of day and day of month.

a compass - provides degrees from north.

a pressure transducer - used to calculate depth

Where the OS itself is written in C, the most immediately useful language is C, and is, as yet, the only language interfaced to the OS. The full capabilities of C language are available as well a as subset of OS calls, such as read and

64

write. Below is a list of the calls currently supported,
followed by a brief description of each call's function.

```
read (fd, buff, nchars)
write  "   "      "
iread  "   "      "
iwrite "   "      "
exit ()
sleep (hundredth)
set-status (name, new-status)
lock (fd, <write-0, read-1>, <1-lock, 0-unlock>)
raw (fd, <1-raw, 0-normal>)
load (name, time, stack-size, <run-0, pend-1>)
```

read (fd, buff, nchars)
```
    int fd;
    char buff [];
    int nchars;
```

      Where nchars characters are read from the channel
designated by fd into buff. Note: the task making this
request will wait until the characters requested have been
read and at that time a count of the chars will be returned.

write (fd, buff, nchars)

```
    int fd;
    char buff[];
    int nchars;
```

      Where n chars are written to the channel designated by
fd from buff. Note: the task making the write request will
wait until the characters to be written are transferred into
the OS's buffer. As with a read request, the actual number
of characters transferred is returned by the OS.

iread (fd, buff, nchars)

iwrite (fd, buff, nchars)

```
    int fd;
    char buff[];
    int nchars;
```

      Both of the routines function in a fashion similar to
read and write except that the task making the request never
waits. The values returned by these routines indicate
whether or not the operation was performed and if performed,
to what extent.

exit ();

      Causes the termination of the calling task, by setting
its status to "FINISHED" and switching it allowing the next
task to run.

65

sleep (hundredths)

    int hundredths;

        Puts the current task to sleep, deactivates it, for a
period of time specified in hundredths of seconds.

set-status (name, new_status)

    char name [];
    int new_status;

        Sets the status of the task specified by name to what
is specified by new_status.

lock (fd, read_write, lock_unlock)

    int fd, read_write, lock_unlock;

        Locks or prevents other tasks from accessing an I/O
channel. The fd specified the channel to be locked, read-
write designates the read (1) or write (0) channel and lock-
unlock indicates whether to lock (1) or unlock (0) the
channel.

raw (fd, raw_normal);

    int fd, raw_normal;

        Sets the mode of the channel specified by fd. The mode
is set to either raw (1) or normal (0). The meaning of raw
mode varies with the channel requested. Note: there is a
certain amount of protection, such that a task must own a
channel to put it in raw mode.

load (name, time, stack_size, run_pend)

    char name[];

    int time;

    int stack_size;

    int run_pend;

        Sets up a task called name which has a stack of stack-
size words, and when running has a time slice of time units.
After the task block is set up the task is loaded through
the camera port. And either run (0) or pended (1) depending
on the value of the fourth parameter.

        In order to make use of the I/O routines the user must know
the correspondence between fds, or channel numbers, and the

physical devices they access.   Below is a list of the current fd
assignments:

    0 - console r/w

    1 - thruster r/w

    2 - NAV r/w

    3 - MBM r/w

    4 - unsupported camera

    5 - compass read only

    6 - pressure read only

    7 - date time r/w

    The set_status routine requires the specification of a new
status for a process.   This new status is designated by an
integer value one to eight. The meanings of the different values
are listed below:

1-INPUT-WAIT When a task has an unsatisfied read request

2-OUTPUT-WAIT When a task has an unsatisfied write request

3-RUNNING   When a task is actually executing instructions

4-WAITING   When a task is waiting to run

5-PENDED   When a task has been stopped during execution or
              when it has not yet been started up

6-ERROR   When an error occurs in the execution of a trap #1.

7-FINISHED   When a task has completed its execution. This
              status is normally set by exit ().

8-SLEEPING   When a task is in a sleep state. When the sleep
              period is up the status is changed to WAITING.

The VOS Monitor

    The VOS monitor is a high level supplement to the SIMS 68000
Monitor V.2.0. As yet it provides none of the memory test/access
routines but does enable the testing and use of the higher level
vehicle functions.   In addition to providing commands for
accessing and testing the various I/O channels it allows for a
certain amount of task control. This control consists of loading
new tasks, starting and stopping ones which have been loaded and
checking their current status.   Lastly there is a command which
allows the testing of the sleep function.

The commands accepted by the monitor are:

        HELP
        LOAD
        DATE
        COMPASS
        PRESSURE
        THRUST
        EXIT
        NAV
        MBM
        SLEEP
        SET_STATUS

The specifics of these functions are contained in the next section.

(1) HELP <cr>

prints the above list of commands

(2) LOAD <name> <time slice> <stack size> <run-0,pend-1> <cr>

        Allocates a process descriptor block (PDB) for the task which is added to a linked list of PDBs and a stack, then it prompts the user with the correct load address of the process. After this has been done the system then waits, reading the camera port for the load module. When the task has finished loading it is either run or pended, depending on the last parameter.

(3) DATE <cr> or

DATE <mins> <hours> <day of week> <day of month> <month> <cr>

        This allows the user to both set the date and query the OS to find out what it thinks the date is. When setting the date all parameters are optional. The list of numbers is interpreted and assigned from left to right. Given this if three numbers are provided they are taken to be <mins> <hours> and <day of week>, leaving the <day of month> and <month> unchanged.

(4) COMPASS <cr>

        When invoked, this command causes the compass to be read continually and its reading to be printed out on the console. Where there is approximately a two second delay in the compass and a delay due to output character buffering, readings being printed are generally several seconds old. A ctl-d typed on the console causes control to be returned to the monitor.

68

(5) PRESSURE <cr>

When invoked, this command causes the pressure to be read continually and its reading to be printed out on the console. As with the readings of the compass, values being printed are generally a few seconds old. A ctl-d typed on the console causes control to be returned to the monitor.

(6) THRUST <cr>

(7) NAV <cr>

(8) MBM <cr>

Each of these commands behave in the same fashion. When invoked each of these causes the console to go into transparent mode with the corresponding computer. A ctl-d typed on the console returns control to the monitor.

(9) SLEEP <hundredths> <cr>

Causes the monitor to sleep for the time specified in hundredths of seconds.

(10) SET_STATUS <name> <new-status>

This explicitly sets the status of the task(s) with a matching name. The status specified is assumed to be a number with the same meaning as described in the code for the operating system "declare.h" and as listed earlier in this paper.

(11) EXIT

At present it does nothing, it could easily be used allow the monitor to be completely shut down during a mission.

7.4 Writing Programs for the Vehicle

Programs running on the vehicle have very few, if any limitations. The primary concern of the author should be the size or memory requirements of the program. Because of this the use of general purpose formated I/O routines should probably ve avoided because of their high overhead. Because programs written in C are the most straight forward to compile, test and load, the procedure for their use will be presented first.

As an example, the process of writing, loading and running a program to run the forward thrusters for 10 seconds and then run them in reverse for 2 seconds and halt will be described. The program should first be written and tested on the ERG system. The following is a version of the program, described above, set

up to run on the ERG system. The primary differences are the fds
or channel numbers used in doing I/O.

```c
{
/* a program  to test the thruster communications and demonstrate
     debugging on the ERG system */

#define THRUST 1

    /*  set the fd so that the output intended for the thrusters
    now goes to the tty */

#define CONSOLE 1

    /* set the fd that the output intended for the console  goes
    to the tty /

main ()          /* the main program can't have any arguments when
                    running on the vehicle */

write(THRUST,"I",1);     /* initialize the thruster computer */

sleep(2);                /* sleep 2 seconds waiting for the
                            initialization to be completed */

write(THRUST,"7F',2);    /* set the thrusters to go forward full
                            speed */

sleep(1Ø)                /* sleep the specified ten seconds */

write(THRUST,"7B",2);    /* set the thrusters to go back full
                            speed */

sleep(2);                /* sleep the specified two seconds */

write(THRUST,"H",1);     /* halt all the thrusters */

write(CONSOLE,"TEST FINISHED\n",14);

                         /* inform the user that the job is
                            done */

}
```

        Once the program has been typed in it may be compiled and
run on the ERG by typing: c test.c; *xeg <cr>.   After printing
some messages concerning the compilation the program will
execute, producing the following output:

                    I7F7BHTEST FINISHED

That essentially completes the testing that may be done on the
ERG system.   Now to run the same program on the vehicle some
changes must be made to account for differences in fd assignments

and the resolution of the sleep function.   Once the changes have
been made the program appears as follows:

```
/* a program to test the thruster communications and demonstrate
   debugging on the vehicle system */

/* to actually run this program and have it start ano stop the
   thrusters the THRUST fd would have to be changed to correspond
   to the thruster computer */

#define THRUST 0

    /* set the fd so that the output intended for the thrusters
       now goes to the tty */

#define CONSOLE 0

    /* set the fd that the output intended for the console goes
       to the tty

/

main ()          /* the main program can't have any arguments when
                    running on the vehicle */

{

write(THRUST,"I",1);       /* initialize the thruster computer */

sleep(200);                /* sleep 2 seconds waiting for the
                              initialization to be completed */

write(THRUST,"7F",2);      /* set the thrusters to go forward full
                              speed */

sleep(1000);               /* sleep the specified 10 seconds */

write(THRUST,"7B",2);      /* set the thrusters to go back full
                              speed */

sleep(200);                /* sleep the specified 2 seconds */

write(THRUST,"H",1);       /* halt all the thrusters */

write(CONSOLE,"TEST FINISHED\n",14;

                           /* inform the user that the job is
                              done */

}
```

The program may now be loaded and run on the vehicle using the following series of steps:

1. compile the program producing the equivalent assembler code by typing cdump test.c <cr>

2. assemble that code by typing *assem test.s <cr>

3. step 2 produces a file called test.o. It must be linked and relocated with the vehicle routines at the proper load address. To find the correct load/relocation address issue the load command on the vehicle computer. The vehicle will respond with the load address. Now copy the file, test.o, to "/usr/rec/monitor". To link and relocate the routine type:

        uline <addr> test.o <cr>

   where addr is the hex address provided by the vehicle O.S. This command produces a file called xeg.

4. Using an ADM with an extension port you may now either load the program directly, or make a tape for loading later. If loading directly, the extension port should be hooked up to the camera port/tether. Since the load command has already been issued the vehicle O.S. is waiting for the load module which may be sent by typing:

        hex -s xeg <cr>

5. Upon completion of the load, the VOS monitor will issue its prompt. If the program was loaded in a run state then it will start execution immediately. Otherwise it will be in a pended state, which can be seen by typing ctl-t.

8.   NAVIGATION SYSTEM

8.1   Description

The   purpose   of the navigation system is to   determine   the
vehicle's location in the x, y coordinate system plane. A block
diagram of the navigation system and a photo are shown in Figures
29 and 30.

This system consists of:

a)   6100 computer and associated drivers with two fields of
     memory.   The memory contains a total of 8000 twelve bit
     words in RAM and ROM.

b)   A supervisor UART to communicate with a terminal and a
     host UART to communicate to the Motorola 68000 command
     computer.

c)   A math processor for calculating position.

d)   An interface card which handles·the switching, timing,
     and   control for the transducers as well as   the
     receiver/detectors.

e)   Three sets of receiver/detector card pairs.

f)   Nine filters, three each at 110kHz, 114kHz, 118kHz.

g)   Three transducers which transmit at 95kHz and receive in
     the range of 100 - 130kHz.

h)   Three transponders which are interrogated at 95kHz and
     which transmit in the present configuration at either
     110kHz, 114kHz or 118kHz.

The   navigation   system operates as   follows.   Initially a
start pulse triggers the selected transducer (either 1, 2, or 3).
The particular transducer is selected in the navigation algorithm
which will be described later. The selected transducer transmits
a 1ms burst of 95kHz signal. Each of the three transponders
responds at a fixed frequency (110kHz, 114kHz, or 118kHz) and at
different fixed delay settings of approximately 51ms, 28ms, and
10ms respectively.   The transponders are prevented from retrig-
gering for a period of 300ms.   The transponder output pulsewidth
is also 1ms wide.

Each of the three transducers on the vehicle receives all
three signals.   The transducers provide a preamp gain of 30db.
The signals are then passed to the receiver cards which provide
nominally another 60db of gain (Figure 31). The signals are then
filtered such that each transducer provides three signals (110,
114, 118kHz) to each detector card. The filters' responses as a
function of frequency are shown in Figure 32.
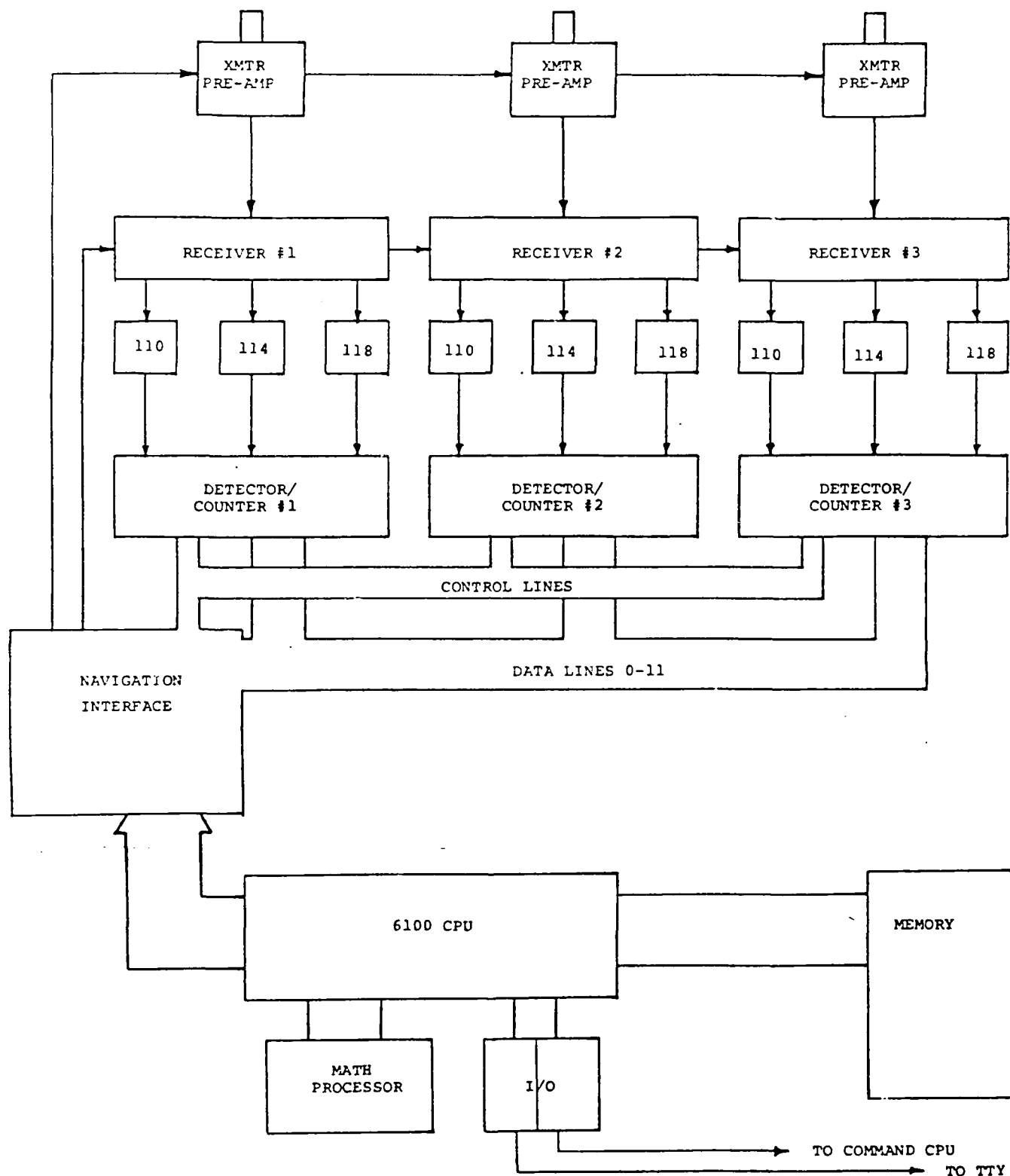
FIGURE 29:   SIMS NAVIGATION SYSTEM
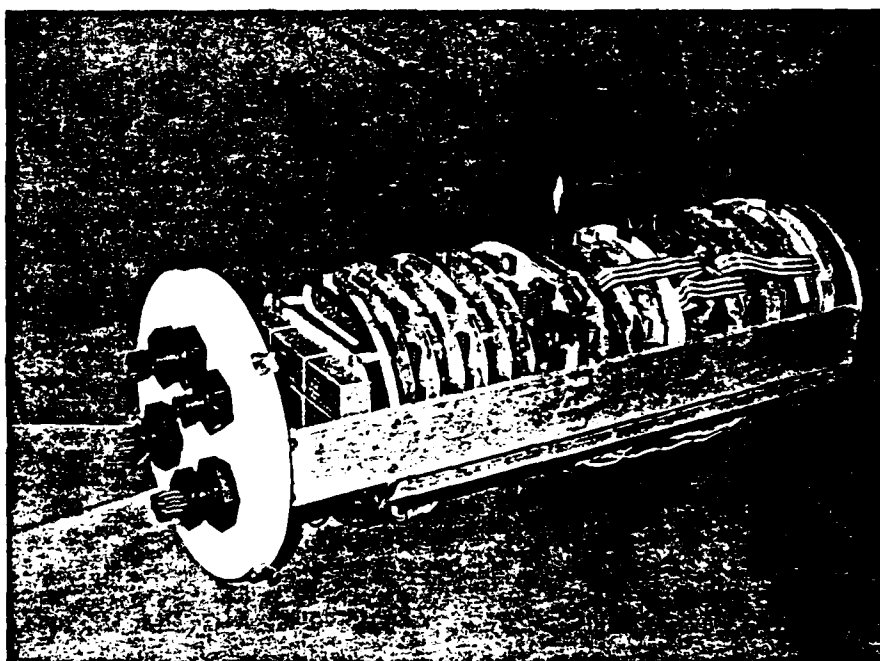
Navigation Computer

FIGURE 30
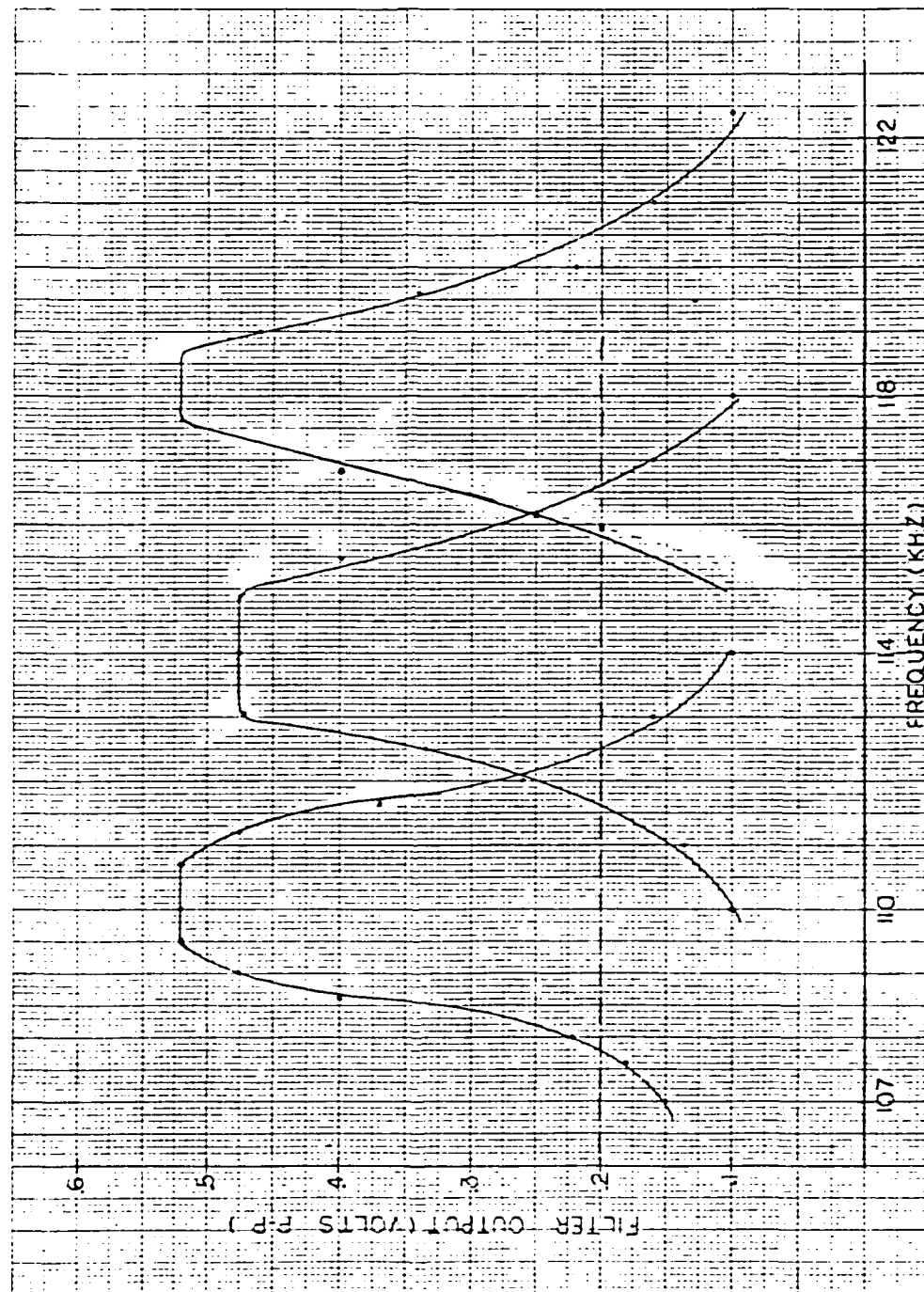
FIGURE 32: FILTER RESPONSE TO 1msec BURST

76
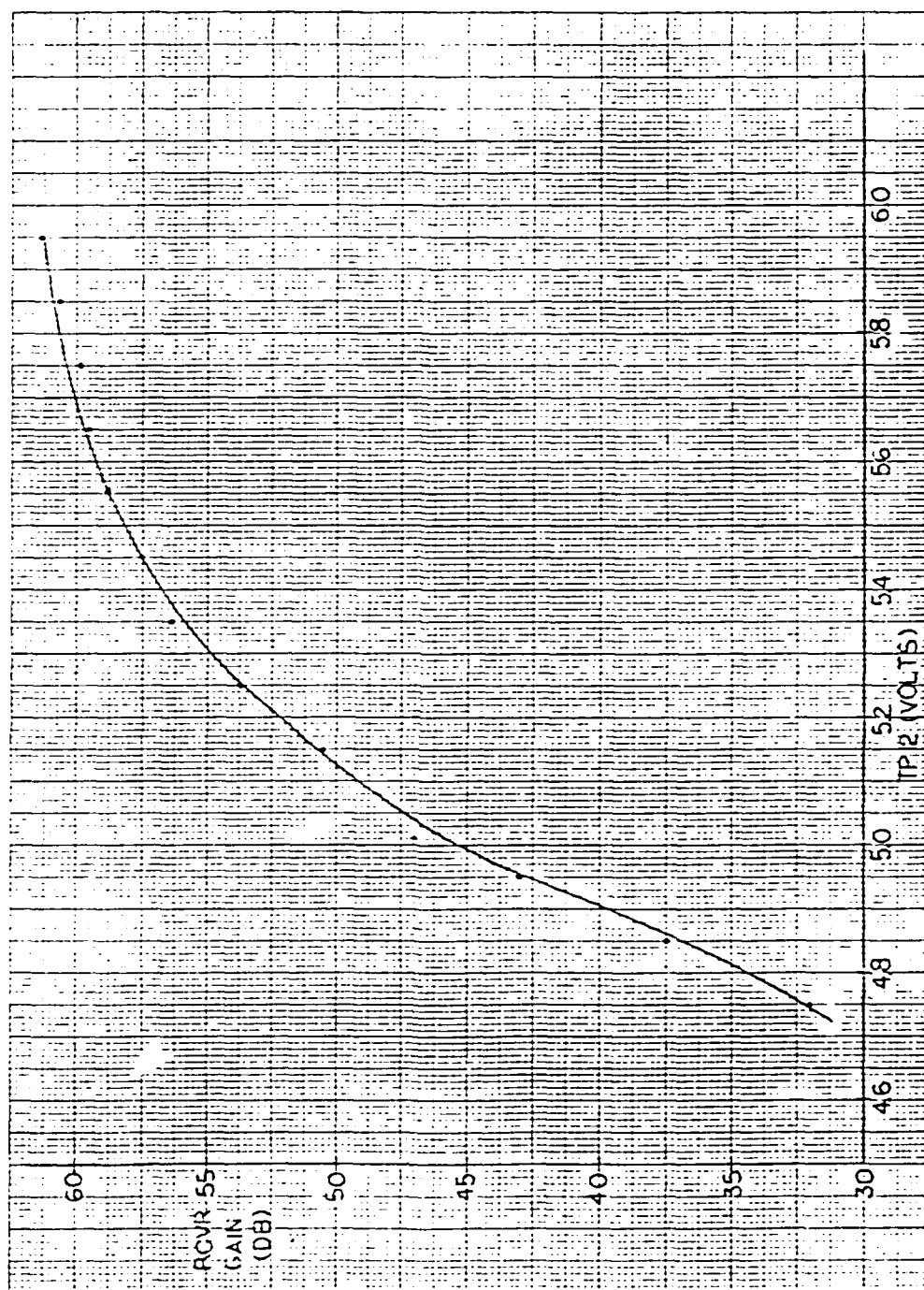
FIGURE 31: RECEIVER GAIN VS. CONTROL VOLTAGE

77

The filters used are SM-WF2 mechanical filters built by SEIKO. They have an insertion loss of 10db maximum. Each filter output is connected to a detector counter circuit. There are nine filters, three for each transducer input, hence there are nine returns to detect and count. The counters are started at the same time as the 95kHz transmit pulse is sent. The counters are stopped when the differential threshold level in the detector circuit is reached. The threshold level is settable and must be adjusted properly to discriminate against the lower level filter outputs from adjacent channel signals. The signals are spaced in time and in frequency. The differential threshold level is set to preclude a detection from the primary signal in adjacent frequency channels. This setting thus controls system sensitivity and hence maximum distance from which a signal can be received.

The counters in the current system are 12 bits long. A plot of the distance between the transponders and receivers is shown as a function of counts in Figure 33. The plot takes into account the various turn-around-time delays of each transponder as evidenced by the different zero crossing points.

It is obvious from this plot that the maximum count of 7777 octal corresponds to a maximum distance of approximately 295 ft, 245 ft and 195 ft for transponders A, B, and C respectively. This is effectively the distance limit due to using the 12 bit 6100 computer system. A 68000 computer (16 bit) system has already been designed and will be used for missions requiring longer range. This system is discussed in Appendix C.

Once the nine counts, corresponding to nine separate and distinct distances are received, the turn-around times are subtracted and nine range values may be calculated. Figure 34 shows the nine ranges corresponding to the counts received. The generalized equation for determining range is given below.

$$R(ft) = (\text{Total count} - \text{TAT}) \, (C \times t) \div 2 \quad \text{in feet}$$

R = Ra1 or Rb1 or Rc1 (ranges corresponding to Figure 35)

TAT = Transponder turnaround time

C = Speed of sound in H_2O for conditions (feet/second)

$t = 33.3 \times 10^{-6}$ sec. (30kHz clock)

This equation results in a one way range distance in feet. It assumes the transmitter location is also the receiver location. When the receiver location and transmitter location are different, then the distance equation becomes:
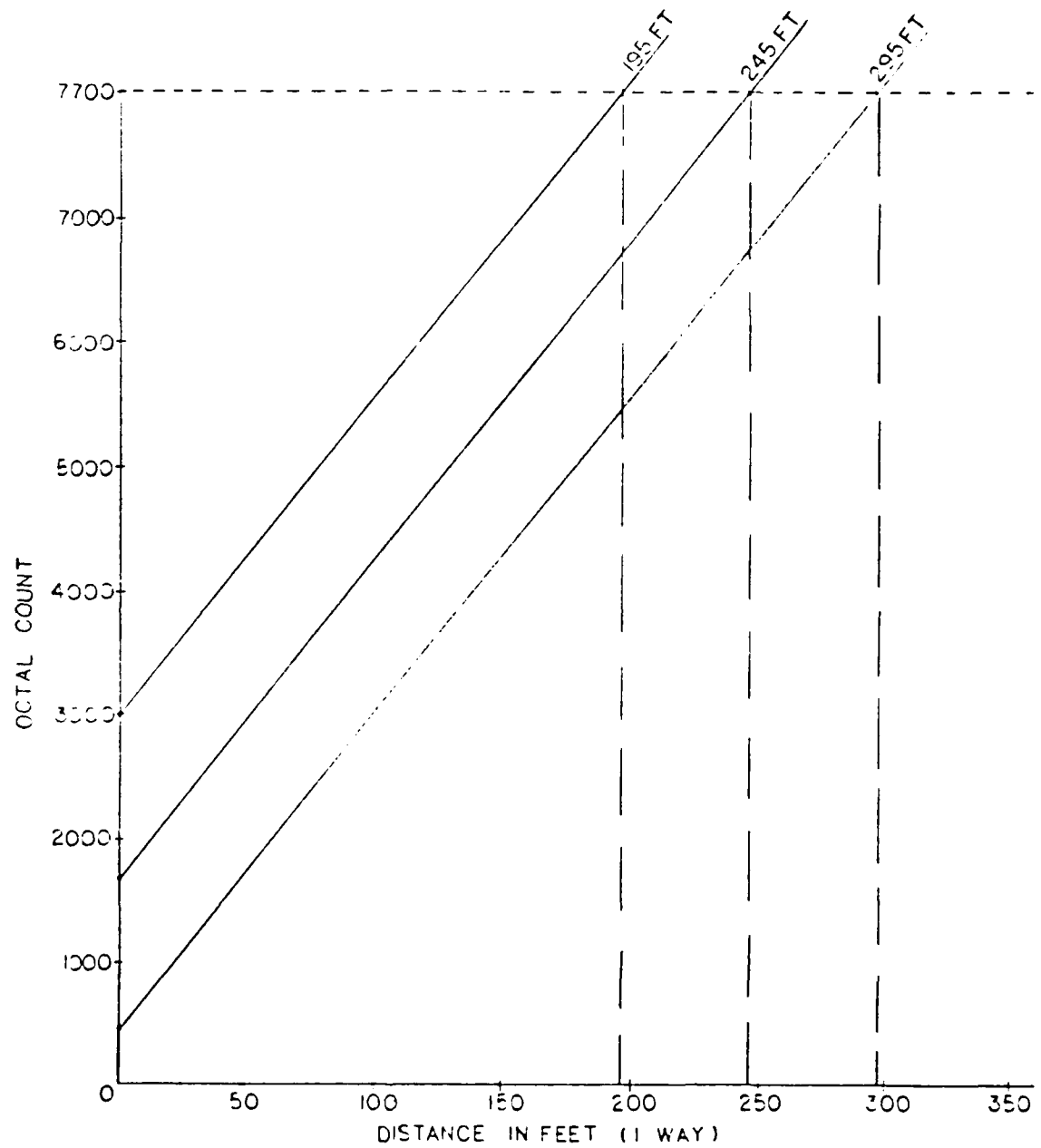
FIGURE 33: SYSTEM RANGE LIMIT DUE TO 12 bit SYSTEM
OCTAL COUNTS vs. DISTANCE

79

DATA FORMAT

OCTAL COUNTS	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
XPONDER	B	A	C	B	A	C	B	A	C
RECIEVER	1	1	1	2	2	2	3	3	3

	B	A	C
1	$X_1 = 2R_{B1}$	$X_2 = 2R_{A1}$	$X_3 = 2R_C$
2	$X_4 = R_{B1} + R_{B2}$	$X_5 = R_{A1} + R_{A2}$	$X_6 = R_{C1} + R_{C2}$
3	$X_7 = R_{B1} + R_{B3}$	$X_8 = R_{A1} + R_{A3}$	$X_9 = R_{C1} + R_{C3}$

ASSUME THAT DELAYS ARE
SUBTRACTED FROM COUNTS

XPONDER $A_X A_Y A_Z$
A
118 KHZ

R_{A2} R_{A3}

R_{A1}

R_{C2} $C_X C_Y C_Z$
XPONDER
C
110 KHZ

FRONT

R_{C3} R_{C1}

R_{B2}

R_{B3}

R_{B1}

XPONDER $B_X B_Y B_Z$
B
114 KHZ

FIGURE 34: GENERAL SYSTEM LAYOUT

80

$$R = (\text{Total count} - \text{TAT} - \text{count for Ral or Rbl or Rcl})(C \times t)$$

in feet

where R is now the one way distance from a given transponder to a receiver that is not transmitting.

Assuming all good returns, we could now know nine range values, one from each transducer to each transponder. We know which transducer sent out the interrogate signal, and we also know the exact position on the vehicle of each transducer.

Referring to Figure 34 again, all that is required to calculate a vehicle position in the superimposed x, y axis are two returns on any one receiver.

The vehicle position calculations are performed in the math processor card. The calculations which are performed by the math processor are listed in Figure 35.

Notice that the range values in this example are denoted as D values, and the horizontal plane range values are denoted by R. Equations 4 and 5 are the generalized form of the solution for the position of a given receiver in the x, y axis. These equations become simplified when the coordinate system is aligned such that two of the transponders make up the y axis and transponder B is located at the origin. Equation 4 in Figure 35 now becomes:

$$Vx = \pm d21$$

and equation 5 becomes:

$$Vy = Ay \pm d11$$

The math processor is the NMOS American Micro Devices 9511. It is not inherently compatible with the 6100 computer, therefore timing circuits and data transfer buffers were incorporated in the design. The math processor also uses significant current (100ma). In order to conserve energy, a switching circuit is used to turn on power to the math processor only when it is calculating.

Once the x, y position of a particular transducer on the vehicle is calculated, it is stored in the 6100 buffer. The 68000 command computer requests this buffer data whenever it needs a position update from the navigation computer. The cycle rate of the navigation computer (2 to 3 times per second) is faster than the overall cycle rate of the command computer (approximately .8 seconds per cycle) hence the command computer is always receiving an up to date vehicle position.

We are currently using a compass to determine vehicle bearing, however, there is enough information available in the navigation data to also calculate bearing and this will be

$$\text{①} \quad R_A = \sqrt{D_A^2 - (V_z - A_z)^2} \qquad\qquad R_{BI} = \sqrt{D_{BI}^2 - (V_z - A_z)^2}$$

$$\text{②} \quad D_{II} = \frac{R_A^2 + D_{AB}^2 - R_B^2}{2 D_{AB}}$$

$$(V_z - A_z) = \text{VEHICLE} - \text{XPONDER}_z$$
$$D_{AB} = 60 \text{ FT} = 377 \text{ DEC. COUNTS}$$

$$\text{③} \quad D_{2I} = \sqrt{R_A^2 - D_I^2}$$

$$\text{④} \quad V_X = A_X \pm \frac{D_{II}}{D_{AB}} (A_X - B_X) \pm (A_Y - B_Y) \frac{D_{2I}}{D_{AB}}$$

$$\text{⑥} \quad V_Y = A_Y \pm \frac{D_{II}}{D_{AB}} (A_Y - B_Y) \mp (A_X - B_X) \frac{D_{2I}}{D_{AB}}$$
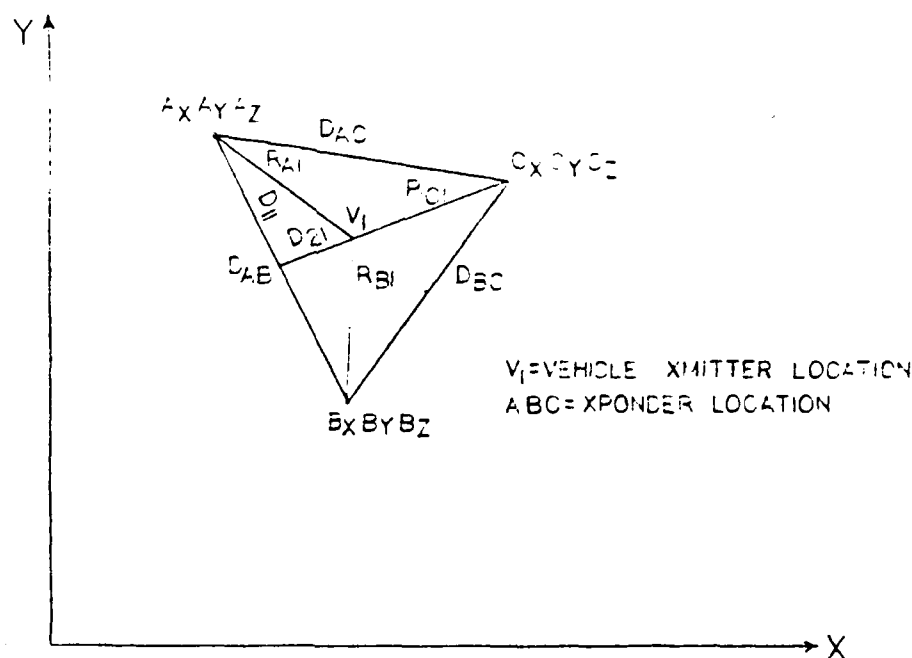
FIGURE 35: NAVIGATION SYSTEM GEOMETRY

82

incorporated in the system. A discussion of bearing calculations, navigation, and sensitivity is presented in Appendix D.

8.2 Test Results of Navigation System

Figure 36 consists of an actual data printout from the vehicle system. The first line of data consists of the x and y position in octal counts followed by a reliability word, followed by the nine octal counts, three from each receiver. The transponder turn around times have already been subtracted from the counts before they are displayed. If a comparison is made between the x, y position presented in the navigation system, and the x, y position calculated from the raw counts displayed in line 1, we find in line 7 and 8 that the position calculations are in agreement in x (29.4 ft vs. 29.6 ft) and y (18.4 ft and 18.6 ft) to within 0.2 feet.

Additionally, a calculation was made to determine how close in position the data indicates the three receivers on the vehicle are. This was done using the data from transponders B and A for each of the vehicle receivers 1, 2 and 3. The results of these calculations are shown in Figure 37. The x, y positions are shown and plotted on an expanded scale. It is shown that the navigation system results indicate that the receivers are separated by 3.0, 3.1, and 3.1 feet respectively. The actual distance between receivers is 3.0 feet indicating that the system is positioning the receivers within .1 feet of their relative location.

It is also obvious from the plot that the vehicle forward axis is pointing to approximately 60 degrees relative to the x axis. The vehicle compass was not working properly at that time and was indicating (Figure 36 line 5) 503 degrees, or 143 degrees after subtracting out 360 degrees. The compass had been a great source of unreliability. Consequently it has been replaced. The new vehicle compass is a solid state flexgate compass type 869 built by Endeco Corporation. It has an accuracy of ± 1 degree and a 10 bit digital resolution of $\pm$.35 degrees. A +5V TTL serial output is used. A 360 degree reading corresponds to 1023 pulses. The compass was mounted on the vehicle and testing performed to measure compass error due to thruster motor effects as well as complete system effects on compass accuracy. It was found that in worst case conditions the readings could be off by approximately 3 degrees. This error is largely due to thruster motors.

It is important to note, however, that at the time the command computer did not know that the compass was erroneous and calculated a new x and new y (lines 9 and 10 in Figure 36). The new x and new y is merely the translation of the x, y coordinate system to the center of the vehicle. It is calculated using the vehicle bearing and the x, y position of the transmitting receiver which in this case was number 1 (line 6, Figure 36). In comparing the navigation system new x, new y (603, 365) to the

VEHICLE NAVIGATION DATA vs. CALCULATED

Line#		Oct X	Oct Y	Rel Word	Receiver 1 114 B	118 A	110 C	Receiver 2 114 B	118 A	110 C	Receiver 3 114 B	118 A
1	RDM:	563	350	6001	711	1215	510	721	1237	503	700	1233
2	TIME: 9505722 (in hundreds of a second)											
3	RELWORD: 6001											
4	PRESSURE: 171 (in inches of depth)											
5	COMPASS: 503 (in degrees) -compass not functional-											

	NAV. SYSTEM OUTPUT		CALCULATED OUTPUT
6	PINGER: 1 = xmitter		PINGER 1
7	X: 562 = 29.4 ft.		X: 564 = 29.6 ft.
8	Y: 350 = 18.4 ft.		Y: 352 = 18.6 ft.
9	NEWX: 603 = 30.8 ft		NEWX: 543 = 28.4 ft.
10	NEWY: 365 = 19.5 ft		NEWY: 336 = 17.8 ft.
11	DELTAT: 10.430		
12	DELTAX: 1.055		
13	DELTAY: 0.479		
14	DELTAZ: 0.192		
15	DELTAB: 0.000		

		X	Y	Z						
16	NAV.	30.805	19.502	10.083	0.084	0.038	0.016	5.636	0.000	6001
	Calculate	28.4	17.8							

FIGURE 36

CALCULATIONS OF EACH RECEIVER

LOCATION DATA TIME 9505722

20 October 1982

RECIEVER 1 = V_{X1} , V_{Y1} = 29.6 FT.   18.6 FT.
RECIEVER 2 = V_{X2} , V_{Y2} = 32.3 FT.   16.7 FT.
RECIEVER 3 = V_{X3} , V_{Y3} = 29.6 FT.   15.6 FT.
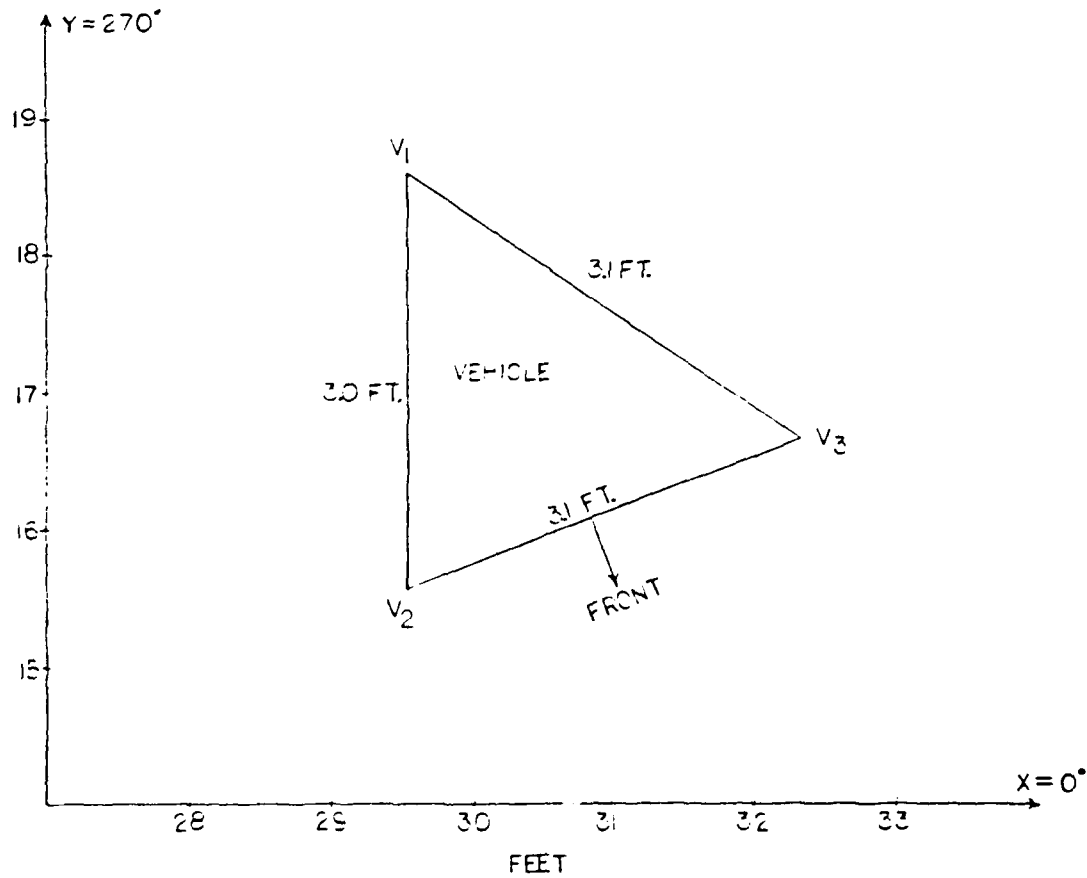
FIGURE 37

expected new x, new y (543, 336) based on an angle of 143 degrees
it became obvious that an error was being introduced. The error
was traced to the software which performs the translation in the
command computer. The software described in Section 15 of this
report was not accounting for the sign (+, -) in the sine
function which it uses. This was corrected, and required very
little software modification.

Figure 38 is a printout of actual vehicle data in a
different format than Figure 36. The raw counts are not
displayed in this figure. This data demonstrates several
important results. The first four lines of data are all
calculated using transmitter 3 as evidenced in the reliability
word 1033 which translates as follows:

1 - a calculation was made

0 - some data dropout (i.e. not all 9 counts were good)

3 - data calculated to transducer 3

3 - transmitter was transducer 3

The vehicle at the time of this printout was positioned at
the entrance to the structure window 1 and was essentially
stationary. A comparison of the variation of x, y position of
the vehicle for the first four lines indicates a maximum
variation (Δ x) in x of .24 feet and in y a variation (Δ y) of
.32 feet. This was taken over a period of 11 seconds total
time. It should also be noted that the correction to the
translation algorithm described above had not been made at this
time and hence the absolute accuracy of the position was not
determined from this data. The data and position stability,
however, for a given transmitter is demonstrated.

The effect of the translation error is also demonstrated
when the transmitter switches such as in line five when
transmitter 2 is used and the initial Δ x goes to 0.4 feet. This
is demonstrated again more emphatically in Figure 39 (a
continuation of Figure 38) line 7 when the transmitter switches
to number 1 and a Δ x of .636 feet and Δ y of 1.43 feet is
produced. Notice that thereafter while transmitter 1 is on
(lines 8, 9, 10, 11) that the x, y position remains stable to
within .08 feet in x and .159 feet in y.

The first seven digit word of each line is the real time in
hundreds of a second. At the time of this test the command
computer cycle time was approximately 3 to 4 seconds. A
component error in the timing hardware of the command computer
was found, and since that time the system cycle rate has been
brought down to approximately 1.5 seconds.

Another significant finding can be shown from the data in
Figure 40. This data is from the vehicle system but the vehicle
was at a different location in the structure. Notice that the x,

STRUCTURE WINDOW 1 DATA

Rel. Word
c rx
a em
l xt
c rr

Bearing
Φ

Line#	Time		X	Y	Z	Z (feet)	ΔX	ΔY	ΔZ
1	8941624	14 10 34 0 0 335 0 1033 .038 0.999 5.846 0.000 1033	14.010	10.905	34.083	0.000 0			
2	8941965	14 10 34 0 0 335 0 1033 .000 0.000 5.846 0.000 1033	14.010	10.905	34.083	0.000 0	0.0	0.0	0.0
3	8942268	13 11 33 0 0 335 0 1033 0.053 -0.083 5.84ʳ 0.000 1033	13.850	11.064	33.833	-0.053	-0.16	+0.16	-0.25
4	8942727	14 10 34 0 0 335 0 1033 0.059 0.054 5.846 0.000 1033	14.089	10.746	34.083	0.052 –	+0.24	-0.32	+0.25
5	8943202	13 10 33 0 0 335 0 1122 0.000 -0.053 5.846 0.000 1122	13.691	10.746	33.833	-0.084	-0.4	-0.0	-0.25
6	8943622	13 10 34 0 0 335 0 1022 0.000 0.060 5.846 0.000 1022	13.612	10.746	34.083	-0.019	-0.08	-0.0	+0.25

FIGURE 38

87

STRUCTURE WINDOW 1 DATA
(continued)

Line#	Time	-θ-	X	Y	Z		Δx	Δy	Δz
7	8944444 14 12 34 1 0 0 335 0 1111 .840 0.040 5.846 0.000 1111		14.248	12.719	34.000	1.489 0	+.636	+1.43	-0.083
8	8944865 14 12 33 0 0 0 335 0 1011 .019 -0.040 5.846 0.000 1011		14.248	12.248	33.833	0.000 0	0.0	+0.08	-0.17
9	8945381 14 12 34 0 0 0 335 0 1011 0.031 0.048 5.846 0.000 1011		14.248	12.099	34.083	0.000 -	0.0	-.159	+0.25
10	8945705 14 12 33 0 0 0 335 0 1011 .025 -0.077 5.846 0.000 1011		14.328	12.179	33.833	0.025 0	+0.08	+0.08	-0.25
11	8945164 14 12 34 0 0 0 535 0 1011 .000 0.054 5.846 0.000 1011		14.408	12.179	34.083	0.017 0	+0.08	+0.0	+0.25

FIGURE 39

88

TECHNOLOGY DEVELOPMENT FOR UNMANNED UNTETHERED
SUBMERSIBLE VEHICLE SYSTEMS(U) NEW HAMPSHIRE UNIV
DURHAM MARINE SYSTEMS ENGINEERING LAB   JUN 84

MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS 1963-A

STRUCTURE WINDOW 2 DATA

Line#	Time			-θ-						X	Y	Z	
1	8989414	13	20	33	2	0	0	338	0	1111	13.134	20.059	33.667	2.268 0
	.777 0.000 5.898 0.000 1111													
2	8989883	12	20	33	0	0	0	338	0	1111	12.975	20.059	33.417	-0.034
	0.000 -0.053 5.898 0.000 1111													
3	8990176	12	20	33	0	0	0	338	0	1011	12.895	20.139	33.667	-0.027
	0.027 0.085 5.898 0.000 1011													
4	8990636	19	22	33	1	0	0	338	0	1011	19.661	22.686	33.667	1.471 0
	.544 0.000 5.898 0.000 1011													
5	8991017	13	20	33	-1	0	0	338	0	1011	13.293	20.298	33.083	-1.671
	-0.627 -0.153 5.898 0.000 1011													

NOTE: Occasional
       Multi

FIGURE 40

89

y, z position is stable except for the fourth line. At this point the x, y position changes by several feet. This seems to indicate that a multi-path count was used in the calculation. Most multi-path occurrences are discriminated against by the detection circuitry, however, on occasion the multi-path does get counted as a signal. This can occur at times using the present strategy of window expansion described in Section 9.2 of this report. In order to correct for this condition an added software filter was written which discriminates against such occurrences based on the vehicle projected envelope of motion for that time sample. It will be made clear during the discussion of the vehicle control algorithm that this is not a difficult task because the command computer has all the information it needs to determine where the vehicle should be during the next time cycle.

8.3 Summary

The current navigation system appears to be capable of distance measurements to within 8 inches and has a stability of 4 inches over time. The translation algorithm was corrected to properly account for sign errors. A software filter was written to eliminate the possibility of gross errors due to occasional multi-path count being used to calculate a position. Additionally a bearing calculation should be added to the software in order to have a redundancy of bearing data. A new solid state fluxgate compass is currently being used in the vehicle system.

9.   6100 NAVIGATION COMPUTER SOFTWARE

9.1   Introduction

The objective of the navigation computer is to provide to the main vehicle control system, the positional information necessary to navigate the vehicle to an underwater structure, and move through or around the structure and return.

The vehicle will be initially placed approximately 100 feet from the target area. The vehicle is equipped with a 3-dimensional numerical map (structure coordinate system) stored in memory containing the structure coordinates of the target, and the positional coordinates of three object transponders (whose purpose is analogous to that of airport beacons). The depth or z coordinate of the vehicle is provided by the main system computer, through a depth sensor.

Software control of the navigation system is through two UARTS, one for the user-TTY and one for the command computer. A resident monitor decodes commands from the I/O ports and perform various selected functions. A list of the available functions may be found in Figure 41.

9.2   Navigation Computer Task Description

During a ping, the counters measure the length of time for the sonar signal to travel from the transmitter on the vehicle (pinger), to the object transponders in the water, the turn around time, and the return time to the vehicle receivers. For convenience and clarity the term "hydrophone" will be used to designate the vehicle receivers and transmitters. Knowing the speed of sound through the water, the distance from each transponder can easily be computed.

Using the coordinates of the object transponders and the respective distances from the transponders, a position in an x-y plane can be computed. When the vehicle is within the 100 ft. range the navigation computer is responsible for vehicle x-y coordinates, absolute heading and their derivatives (x velocity, y velocity, heading dot).

The scenario as so far stated is idealistic. There are physical problems that exist when using sonar in water. There are problems such as multi-path where a signal bounces off the surface or the floor, thus increasing the distance of the sonar travel time. Shadowing where an object blocks the signal entirely, as well as sound velocity in water errors (this varies due to change in temperature, density, salinity, etc.), and errors in placement of the object transponders are all problems which could produce incorrect data. Therefore, before any calculations can be done, the corrected raw data (corrected from two way travel time) must be checked for validity. This is done by comparing the raw data with a window, which is a predicted

NAVIGATION COMPUTER AVAILABLE FUNCTIONS

Description:

This is the command scanner for the navigation computer.
It will be vectored to upon input or output. Will decode
the command and jump to the proper routine.

The Commands Are:

```
/N.   -- Set current field to N
/Carriage Return -- Close current location
/Slash -- Open current location and type contents
/Line Feed -- Close location and open the following location
/NNNNG -- Transfer program control to location specified
/          by NNNN.
/C [lear buffer] (400 - 3777) octal
/D [ump to tape]
/L [oad to memory]
/R [ecord #]
/S [peed of O/S] (see rewait) (init to 7777)
/P [ing transponder] <0> = off, <1> = on
/H [oldoff time] (initially 7000)
/N [umber of records]
/K [ill printout] <0> = off, <1> = on
/E [cho keyboard] <0> = off, <1> = on
/I [nitialize O/S]
/A [ctivate nav system] <0> = on, <1> = off
/Z [coordinate input]
/B [lock data output]
/M [ath processor off]
/X [ducer #] <1, 2 or 3> (initially 1)
/O [dt]
/T [ransfer rdm]  ( <1> = trfr corbuf to rdm)
/J [ump around fld 0 output] <0> = print
```


FIGURE 41

value based on history. By adjusting the width or size (tolerance) of the window a small range of variance can be provided allowing for small errors and displacement from real time.

Initialization of the windows at the beginning of operation is done by obtaining a number of returns and checking for uniformity. When enough data is consistent, a window is formed using the most recent sample. At a large range all transponders may not be heard. In such a case, only windows for data received are to be initially set. Windows not set are initialized directly on the fly as the returns begin to be received.

Since the windows must change as the vehicle moves, they must be continually updated, and since the windows are based primarily upon the previously received sample then there must be a way to determine if a window has been set correctly. In order to detect a faulty window, a coded matrix was devised. The contents of the coded matrix is based upon the element by element comparison between the raw data in matrix form and its corresponding elements of the windows in matrix form, and window sizes in matrix form. If a raw data term is within its window and window size then it is termed as a "good return". If the return is outside of the window and window size, it is termed a "bad return". An element for which no return was obtained is termed a "no return". If a certain element is consistently a bad return then the window may be in error.

If this occurs, then the window size is expanded by a predetermined amount each time a bad return is received until either a good return is received or the window size exceeds an acceptable maximum. If a good return is received, the window size is reset to the minimum value to preserve the maximum filtering effect. If the window is expanded beyond a maximum size considered to be acceptable the window is reinitialized [by a method to be determined (i.e. some form of past history projection, or on the fly)].

Beyond the use of the coded matrix towards window adjustment is the ability to determine what information can be computed. The mathematics used to compute position uses the horizontal distance from two object transponders, which yields two solutions mirrored about a line drawn through the two transponders. To disambiguate, either history (previous samples) or the distance to the third transponder must be used. If the distance to the third unused transponder is available, then by computing that distance twice using the two solutions, it can be determined which solution is closest. If history is available, then by comparing the two solutions with history, the closest one can be determined. If history is available, then upon examination of the coded matrix if there are at least tow "good returns", a position can be calculated. If no history is available then three good returns must be received.

A decision as to which hydrophone to ping is made using the coded matrix. If enough data is received using the current pinger to compute a position, it remains unchanged. When this is not the case, however, a new pinger is chosen by checking the rows of the coded matrix to determine which row (hydrophone) has the most good returns.

If all the returns in a column are consistently bad or no return status even after window correction, then failure of either the counters or the transponder is indicated. If the same conditions exist for a row, then a failure of the hydrophone is indicated. The coded matrix allows window correction, "one-look" information availability, and pinger decision.

This is the theory upon which the algorithm operates. Following is a step by step description of the algorithm from which one may gain a more concise image of the navigation.

9.3 Navigation Algorithm

The navigation algorithm consists of eleven modular segments. The modules are divided and designated according to the function they perform. Modules I and II are initialization procedures are one pass, while modules III and XI are organized in a linear loop with one subroutine called in module III. An explanation of each module follows.

Module I is the system initialization. The coordinates of the object transponders and any predesignated parameters are loaded into memory. All flags used through the algorithm are cleared.

Module II is the navigation computer initialization and window formation. The function of this module is to choose which hydrophone to ping, and to initialize the windows.

When the vehicle has been placed in the water, before it starts moving, each of the three hydrophones are consecutively pinged. The hydrophone with which the most returns are received is designated as the pinger. In case of a tie, the hydrophones have an ordered priority.

Control is passed into the loop starting with module III. The main system is signaled through the reliability word that initialization is over and the vehicle can begin motion.

Module III, the active navigation and data handler. First the "get data" subroutine is called. The "get data" subroutine controls the actual hardware to reset the counters, ping the chosen pinger, start the counters, test for counter overflow (no return), and load raw data (counter values) into the raw data matrix.

After the "get data" subroutine returns with the raw data matrix, the raw data is compared with the window and window size matrices and the coded matrix is loaded accordingly. A "G" for a good return is coded as a five, a "B" for a bad return is coded as a three, and an "N" for no return is coded as a one, thus yielding a unique sum for any combination of returns in a row or column. When the coded matrix is completed control is passed to module IV.

In module IV, the computation decision module, the existing conditions are examined, and decisions are made to determine what information can be calculated. First the range is found to determine which information should be calculated. The coded matrix is examined and the most desirable pinger is chosen based on number of returns received in a row and previously set priorities (pinger one before pinger two before pinger three). If the previous pinger sum is good (two "G's" or more), then the criteria for position computations are checked. Two "G's" in the pinger row and history (last flags) or three "G's" in the pinger row (three flags) are the only situations where position can be computed. If position is computable, then the position flag is set and the last position flag (history availability from module XI) is checked.

Module V, position calculations, checks the position flag to determine if position is to be calculated, if so, the three flag is checked to determine which method of disambiguation is to be used. Once these flags are checked, appropriate action is taken and control is passed to module VI.

Module VI, result processing and window reset, checks and resets the windows. If there are any "G's" exiting in a column, then the windows for non-good returns in that column are based on the good returns with expanded sizes. The windows of any good return are immediately updated with the actual received return. Therefore, if a window is received for which no window exists, then it has been coded a "G" in module III and the window is automatically initialized in this module. If there are no "G's" existing in a column, then the windows remain the same but the window sizes are increased. If the window sizes are expanded beyond a given limit then the windows are cleared and will be initialized "on the fly" as previously explained.

If the position flag has been set, then there are at least two "G's" in a row in which to base the windows for the column. If the three flag is not set, then the third window can be computed upon which windows in that column can be based.

Module VII, the reliability word formation checks flags set in module III to determine which information will be sent to the main system computer. Bits are set in the reliability word which indicate the availability of this information. If the vehicle is within one hundred feet and the position flag is not set, then the history is checked. If history exists a bit in the reliability word is set to indicate old information. If history

95

also is not available, then the old information bit is set, and bits are set for any history that does exist.

Module X, information loading and transmission loads the appropriate information as indicated by the previously set reliability word into the proper memory addresses and sends this information to the main computer.

Module XI, flag processing and history manager checks flags set in module IV and loads the present information into history while setting the corresponding history availability ("last") flags. After history has been updated, all flags are cleared and control loops back to module III.

10. MAGNETIC BUBBLE MEMORY SYSTEM

10.1 Description

The purpose of the magnetic bubble memory system is to store essential vehicle data during a mission. The data stored in the bubble is such that a reconstruction of the mission and post-mission analysis is possible. The data stored in the bubble can be stored in various ways as we will discuss later. For the SIMS mission, it was decided that the following information would be stored for each complete system cycle (i.e. approximately every second):

1 - Exact time in hundreds of a second

2 - 9 counts detected in Navigation System

3 - x, y position calculated in Navigation System

4 - Pressure corresponding to depth

5 - Compass reading or bearing

6 - Translated x and y positions (center of vehicle)

7 - Polarity and speed commands issued to thruster motors

8 - Reliability word

This would correspond to 42 bytes per cycle. Using a single bubble element which has the capability of storing 1.024 Megabits of data, the system is capable of recording the above information at one second intervals for a period of 50 minutes. Since the SIMS mission duration is expected to be approximately 10 minutes, this is more than adequate. For other applications, the amount of data stored and its file structure and data rate would probably be much different. These differences are easily handled with the present system.

The file structure and software that we have designed is discussed in detail in Section 11. Basically the file structure is made up of 20 files, each capable of storing 50 blocks of data. A block of data is defined in our system as 128 bytes. Any file can be selected at any time.

The basic reason for using a bubble memory system in the EAVE vehicle is that it is non-volatile, transportable, and has random access capability.

A block diagram of the magnetic bubble memory system is shown in Figure 42. The system consists of a 6100 CPU, 4,000 12 bit words of memory, a supervisor UART to communicate to a terminal, a host computer UART to communicate with the 68,000 command computer, an interface card to provide control and timing signals for the data transfers to and from the bubble, and
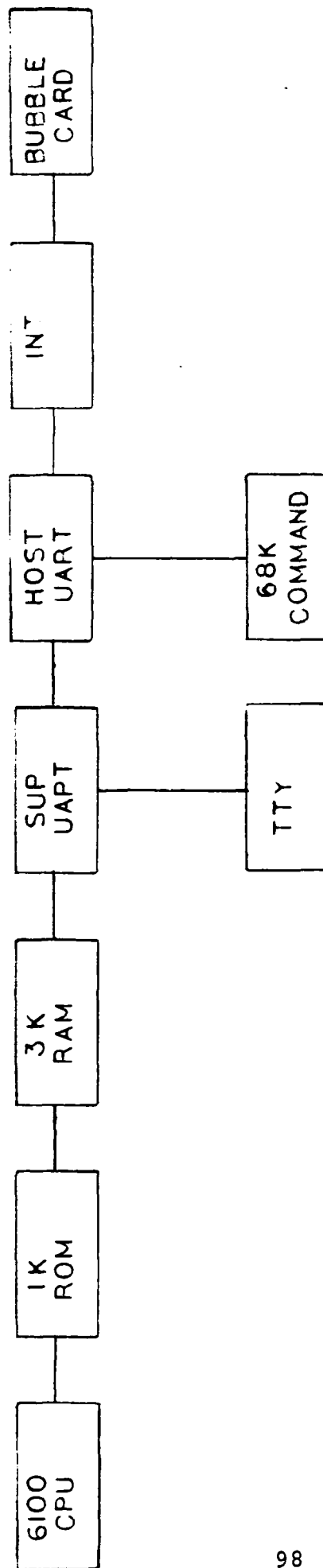
FIGURE 42: MBM SYSTEM

finally the bubble card itself which is an IMB-72 built by Intel Corporation. The physical system is shown in the photograph in Figure 43.

The 6100 computer basically only has to communicate with the 7220 controller (Figures 44 and 45) on the IMB-72 board, and this is accomplished through the interface card.

The interface card to the bubble is made up of an 8 bit bidirectional data bus, an address line Ao, a chip select line, read and write control lines, 2 interrupt lines and a reset line. The heart of the interface is a 6101 Pie element which provides most of the timing and control. A 4 MHz crystal clock is also on the interface card and provides the basic timing to the bubble.

Communication to the BMC (bubble memory controller) is achieved through the user accessible registers. There are three groupings of these registers (Figure 46); the command register, the register address counter (RAC), and the status register.

Most operations or transfers with the bubble involve first writing the RAC with the parameters necessary for the bubble to perform given commands. The parameters which the RAC expects prior to most commands are as follows, and in the following sequence:

1 - Utility register - spare register not normally used.

2 - Block length register LSB - designates the number of bubble pages to be transferred. This number plus the first three bits of the BLR-MSB determine total number of pages. A page contains 64 bytes.

3 - Block length register MSB - the four higher order bits determine the number of FSA (Formatter Sense Amplifiers) used in the system. Each bubble has two FSA's associated with it. For a single bubble system the BLR-MSB four most significant bits would be:

```
7   6   5   4
┌───┬───┬───┬───┐
│ 0 │ 0 │ 0 │ 1 │
└───┴───┴───┴───┘
```

4 - Enable register - provides the user with (a) the capability of using various error correction and detection schemes, (b) interrupt mechanism for data transfer and status, (c) the ability to specify the data transfer data rate and (d) to write the system bootloop (map of bubble) if required.

5 - Address register - LSB - combined with the first three bits of the address register MSB specifies the starting address of the transfer. This starting address in combination with the block length register specifies what and how much data is to be transferred on the next
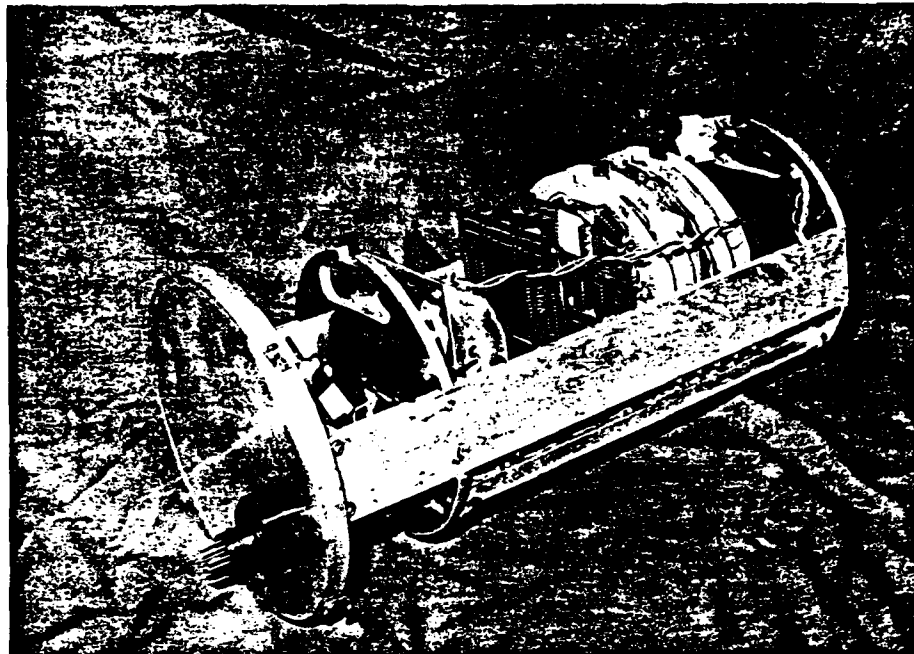
MBM Computer System
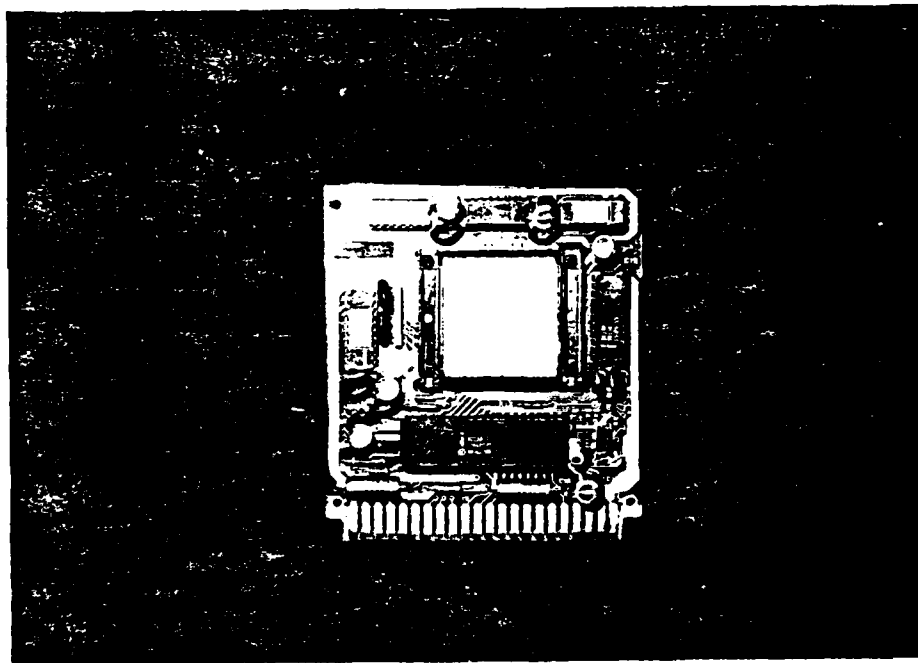
Bubble Memory Card

FIGURE 43
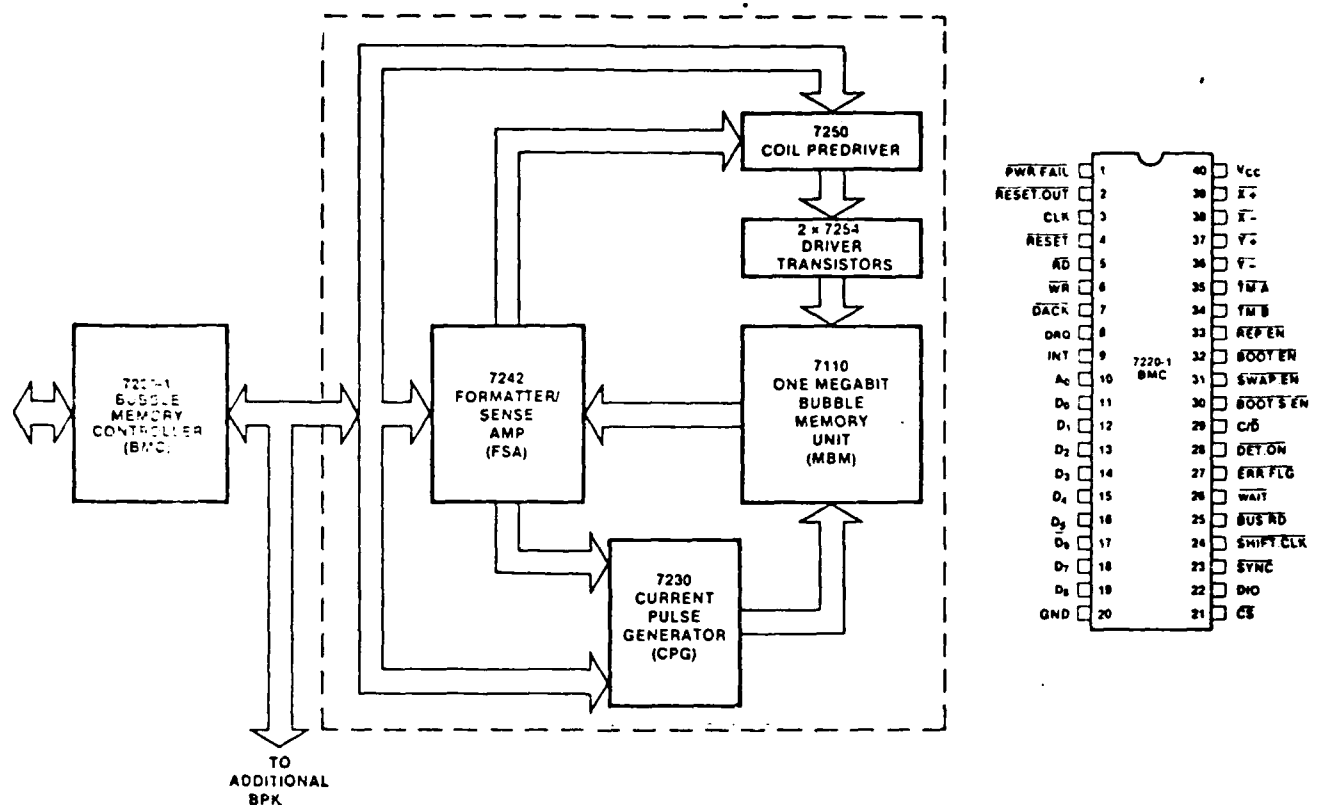
FIGURE 44: 372 BLOCK DIAGRAM
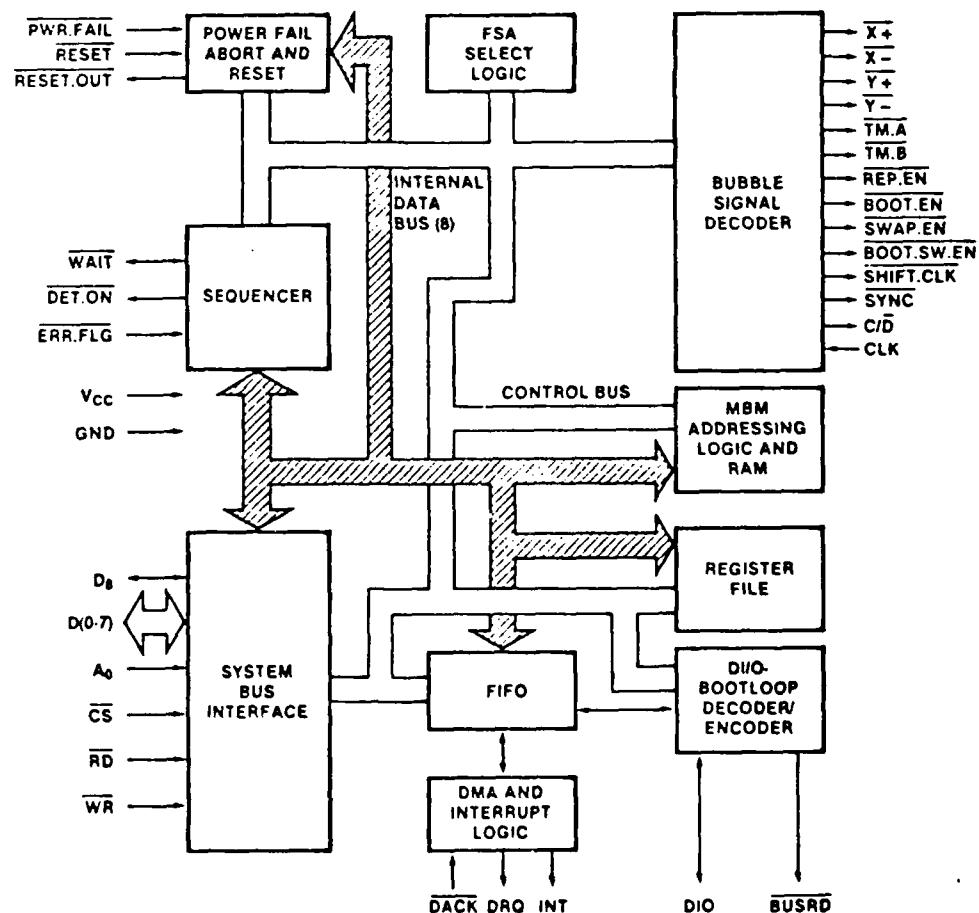
7220 BMC Block Diagram

FIGURE 45

101

Command Code Definitions

D3	D2	D1	D0	Command Name
0	0	0	0	Write Bootloop Register Masked
0	0	0	1	Initialize
0	0	1	0	Read Bubble Data
0	0	1	1	Write Bubble Data
0	1	0	0	Read Seek
0	1	0	1	Read Bootloop Register
0	1	1	0	Write Bootloop Register
0	1	1	1	Write Bootloop
1	0	0	0	Read FSA Status
1	0	0	1	Abort
1	0	1	0	Write Seek
1	0	1	1	Read Bootloop
1	1	0	0	Read Corrected Data
1	1	0	1	Reset FIFO
1	1	1	0	MBM Purge
1	1	1	1	Software Reset

RAC

A0	B3	B2	B1	B0	Symbol	Name of Register	Read/Write
0	1	0	0	0	UR	Utility Register	Read or Write
0	1	0	0	1	BLR LSB	Block Length Register LSB	Write Only
0	1	0	1	0	BLR MSB	Block Length Register MSB	Write Only
0	1	0	1	1	ER	Enable Register	Write Only
0	1	1	0	0	AR LSB	Address Register LSB	Read or Write
0	1	1	1	0	AR MSB	Address Register MSB	Read or Write
0	0	0	0	0	FIFO	FIFO Data Buffer	Read or Write

B3B2B1B0 = 4 bit contents of RAC at the time the user makes a read or write request with A0 = 0

LSB = Least Significant Byte
MSB = Most Significant Byte

BLOCK LENGTH REGISTER MSB   BLOCK LENGTH REGISTER LSB

NUMBER OF FSA CHANNELS (NFC) X NUMBER OF PAGES TO BE TRANSFERRED

ENABLE REGISTER

- INTERRUPT ENABLE (NORMAL)
- INTERRUPT ENABLE (ERROR)
- DMA ENABLE
- MAXIMUM FSA-BMC TRANSFER RATE
- WRITE BOOTLOOP ENABLE
- ENABLE RCD
- ENABLE RCD
- ENABLE PARITY INTERRUPT

ADDRESS REGISTER MSB   ADDRESS REGISTER LSB

MRM SELECT   STARTING ADDRESS WITHIN EACH MBM

Status Register

7	6	5	4	3	2	1	0

- FIFO READY
- PARITY ERROR
- UNCORRECTABLE ERROR
- CORRECTABLE ERROR
- TIMING ERROR
- OP FAIL
- OP COMPLETE
- BUSY

Address Assignments for the User-Accessible Registers

A0	D7	D6	D5	D4	D3	D2	D1	D0	Symbol	Name of Register	Read/Write
1	0	0	0	1	C	C	C	C	CMDR	Command Register	Write Only
1	0	0	0	0	B	B	B	B	RAC	Register Address Counter	Write Only
1	S	S	S	S	S	S	S	S	STR	Status Register	Read Only

SSSSSSSS = 8 bit status information returned to the user from the STR
CCCC = 4 bit command code sent to the CMDR by the user.
BBBB = 4 bit register address sent to the RAC by the user.

FIGURE 46

102

command. Notice that it is possible to transfer as little as one page (64 bytes) at a time, or as much as a bubble can handle (128,000 bytes) in one transfer.
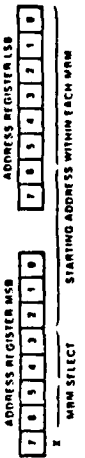
6 - Address register - MSB - bits 3, 4, 5 and 6 specify which bubble is selected in a multi bubble system. This number in conjunction with the BLR-MSB bits 7, 6, 5, 4 specifies the number of FSA channels and hence bubbles in use.

As the above six parameter bytes are passed to the BMC, the BMC auto increments the register address counter locations. After the sixth byte is placed in the counter the auto incrementer is addressing the FIFO.

The FIFO is a 40 byte data buffer through which all data passes to or from the bubble memory. It allows transfer in an asynchronous manner and eliminates some of the timing constraints. As we discuss shortly, the user (6100) must still keep up with data rates specified in the enable register.

Once the RAC has been written, the user then would normally send a command (Figure 46) and then either poll status or await an interrupt from the BMC depending on the way he has set up his parametric registers.

The following is a description of how the MBM system writes a block of data to the bubble. The read sequence is nearly identical.

First the RAC is addressed and the following parameters are sent.

UR	= 0	Not using utility register
BLR LSB	= 00000010	Two page transfer (i.e. 128 bytes)
BLR MSB	= 0001X000	2 FSA channels = 1 bubble
Enable register	= 00001001	Provides for normal interrupts when command is completed (INT) or when FIFO has 22 bytes of space available (DRQ) for transfer. Specifies transfer rate of 12.5 Kbytes/second for write command.
AR LSB	=	Whatever the particular start address is supposed to be.
AR MSB	= X0000	remainder of start address 1 MBM in system don't care

Next the command register is addressed and the initialize command (1) is sent.

The BMC in response to this command, will determine from the parametric registers how many FSA's are present. In this case there are 2. It then reads the bootloop from the bubble. The bootloop is a map of where all the good loops in the particular bubble are located. The map of each MBM may be different. The BMC stores the bootloop in the FSA's bootloop register.

After the initialize command is completed (or any other command) a +5 volt signal is sent on the INT (interrupt) line back to the 6101 Pie element on the interface card. This signal is the indication that the operation is complete.

Next the Reset FIFO command (1101) is sent in order to clear the data buffer before transfer. It too sends an INT signal after completion.

Now the Write Command (0011) is sent. The 6100 then reads the status register until it detects a busy bit. The BMC is now waiting for data and will not transfer anything to the bubble until at least 2 bytes are sent to the FIFO.

The 6100 now sends 20 bytes to the BMC and monitors the DRQ line. (An interrupt line that indicates that there is room in the FIFO for at least 22 bytes). When the line goes high the 6100 sends the next 22 bytes and on the next interrupt it sends 22 more bytes. This is a total of 64 bytes (1 bubble page) but since we have specified a 2 page (128 byte) transfer the system loops back and sends 64 more bytes in the same manner. It is important to note that the 6100 must keep up with the transfer otherwise timing errors will occur. Making use of the interrupt structure and sending 20 or 22 bytes at a time makes the system timing much easier to control. The 6100 computer is operating with a 3.57 MHz crystal and the BMC is transferring data at a 12.5 Kbyte rate. The 2 page transfer takes approximately 10 ms, not counting the random access time vehicle is approximately 50 ms.

Once the transfer is completed the BMC sends a signal on the INT line indicating a complete operation.

It should be noted that once the 6100 begins its transfer of data to the BMC it should not be interrupted by the command computer (and it is prevented from doing so in software) or a timing error occurs.

10.2 Results/Recommendations

The bubble card requires a regulated +5V and +12V supply, and draws 1.92W and 4.8W respectively at 100% duty cycle (constant writing or reading). The stand by total power is typically 1.5 watts total. In order to conserve power a sleep

circuit has been designed and added to the system. This effectively removes power from the bubble except when a transfer of data from the 6100 to the bubble is necessary. Power usage then becomes proportional to the duty cycle of the data transfer. The duty cycle in turn is proportional to how often we are to store (in the present scheme) a block of data (1000 bits). Assuming a duty factor of 5%, the power usage would be less than 0.3 watts. For missions in which data is sampled at low rates over long periods of time the power usage could be orders of magnitude less than this.

The system as it now stands can reliably store data in any file selected. We have not detected any errors in the data thus far.

The current system can be easily expanded either by additional IMB-72 cards, each of which is capable of an additional 1 Mbit of storage, or more simply by substituting the new Intel 4 Mbit bubble chip. This new unit would require little hardware modification and only minor software changes. The new 4 Mbit bubble is actually a smaller physical size than the 1 Mbit unit. Using the 4 Mbit bubbles the physical system currenting is use would be capable of 8 Mbit storage.

11. MAGNETIC BUBBLE MEMORY COMPUTER SYSTEM

11.1 Description

The bubble memory recording system is comprised of two subsystems. The first subsystem is the actual bubble memory data recorder, which is a stand alone computer system. The second subsystem is a host computer that communicates with the bubble computer over a serial link. The host computer sends data to the bubble computer which then handles the filing and access particulars required to store data in the bubble device. This basic configuration may be seen in Figure 47.

The magnetic bubble memory data recorder presently consists of a 6100 microprocessor, two UART cards, one magnetic bubble memory card, and an interface card. Controlling and servicing this system is the bubble memory operating system. The operating system is divided into the following sections:

SCH: Supervisor Channel Handlers
HCH: Host Channel Handlers
BCH: Bubble Channel Handlers
CDS: Command Decision and Scheduler
FSH: File System Handlers
COM: Command Decoding routines

To increase throughput all I/O is buffered. The supervisor channel provides an operator limited access to the bubble system. The host channel is the communication link to the host computer. The bubble channel is the communication link between the 6100 microprocessor and the actual bubble device.

The SCH provides interrupt driven buffered access for an operator into the bubble recorder system. Commands are entered at a terminal and buffered until the command is terminated with a carriage return or the buffer size is exceeded. The buffer is then passed to the CDS for error checking, and command decoding and processing. The first character is the desired command followed by (optional) data. The current legal commands are:

C onfuse   simulates reception of host computer command
D irectory   prints the directory
E rase   erases the directory
O   zero all bubble memory locations
R ead   reads a specified bubble block to terminal

An SCH command may be aborted at any time by typing control-U. The above commands comprise a vestigial list of a longer list of possible file system commands. The above five commands are for testing and monitoring purposes. A lack of available RAM for the bubble computer coupled with the autonomous nature of the mission made the development of a rich and extensive command language for the supervisor channel unnecessary.
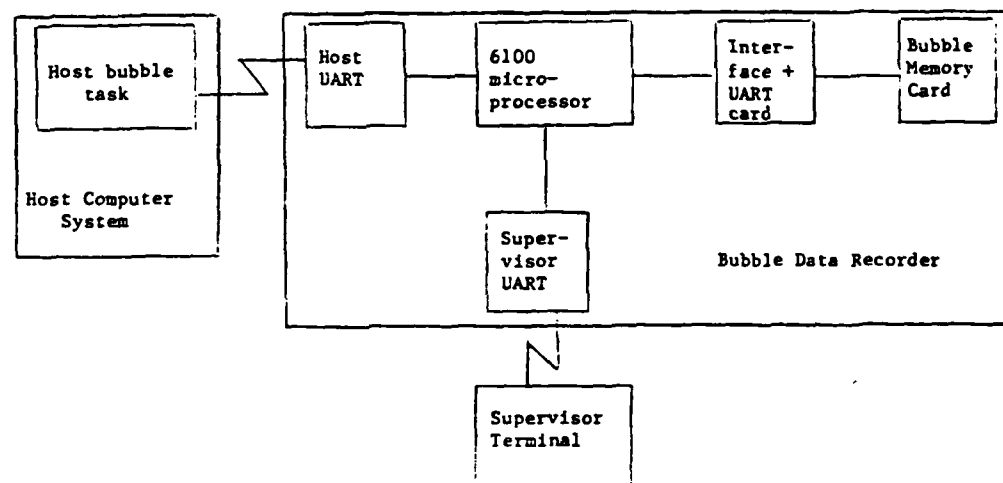
BUBBLE MEMORY DATA RECORDER SYSTEM CONFIGURATION

FIGURE 47

The HCH software is quite different from the SCH although the hardware addressed by each is virtually identical. This difference is due to the protocol structure imposed on all communications between the host and bubble computers. The protocol is based loosely on the high level data link standard x.25-HDLC. There are six separate fields within any one communications packet.

(1)	Flag	signals the start of a packet
(2)	File Number	target file's number
(3)	Control	command to execute
(4)	Count	number of bytes in Data field
(5)	Data	actual data
(6)	Checksum	a checksum of fields 2, 3, 4, and 5

This protocol allows for variable sized packets. The data field may range from length of zero to a maximum of one bubble block (128 bytes). The HCH disassembles the packet as it comes in from the host and if the checksums agree, passes the disected packet on to the FSH for processing. A checksum error will cause the bubble computer to send a not acknowledged/retransmit packet to the host. Successful reception of a packet from the host invokes an outgoing acknowledged packet to be sent back to the host. It is the HCH software that calculates checksums for the packets.

The BCH communicates with the bubble card via the interface card. This software contains the routines necessary to handle the hardware on the bubble device card. Any access to the bubble must be accompanied by various initializing and addressing operations to the bubble memory controller chip. It is the BCH that handles these hardware specific considerations.

The CDS is the main station keeping loop and command decoder/processor processor. The CDS is the "beginning" of the bubble operating system. At start up, all UARTS are initialized, the bubble is initialized, and the file system is initialized by interrogating the bubble for a directory. Control then falls into a status polling loop that monitors a flag. The flag is altered by either the SCH (on detection of a carriage return, end of input buffer, or control-U) or the HCH (reception of a complete packet from the host). Since the SCH and HCH perform interrupt driven input, the CDS is oblivious to the filling of the buffers until the buffer is completed. The value of the flag causes the CDS to execute one of two command scanners; one for the supervisor commands, the other for host commands. The supervisor command scanner calls the proper command in COM or issues an error message appropriately. The host command scanner (actually in the FSH) calls on the filing system to process the host request. If the host request is erroneous, a not acknowledged packet is returned. Once the proper action is completed, the flag is cleared to normal status and control returns to the main station keeping loop.

The COM section contains the code for performing the five supervisor commands.

The FSH is the code for maintaining the file system. This section of the operating system is presently composed of eight routines, the primary one being the reader/writer routine. This is the lowest level primitive and is employed by the other FSH routines whenever bubble access is required. As with a disk, all data transfers to and from the bubble are done in units known as blocks. The reader/writer primitive makes heavy use of the BCH hardware routines to gain access to the bubble. The FSH manages the file system by imposing a specific format on the bubble device.

11.2 Bubble Chip Format

The bubble chip is divided into 2048 pages, each page having a length of 64 bytes. The pages are numbered 0 through 2047. The unit of access to the bubble is in groups of 2 pages called blocks. This provides for 1024 blocks per bubble device; numbered 0 through 1023 and each having a length of 128 bytes. The blocks are accessed using the page address of the first page in the block. This causes legal block addresses to be even numbers in the range 0 through 2046 inclusive. Block numbers and page addresses have the following relationship:

block # = page address / 2

page address = block # * 2

The first 21 blocks (blocks 0 to 20) hold bookkeeping information. The remaining blocks, numbers 21 through 1023, are data holding blocks. Block 0 is the Chip Directory. This block holds information about bubble usage and pointers to other information about each file. This other information is stored in blocks 1 to 20. An up to date copy of the Chip Directory is kept in the 6100 at all times. (See Figure 48).

The first 8 bytes of the Chip Directory contain bubble usage information (Figure 49).

```
Block #        Page Addr.          Block Structure

  0               0          | page 0        page 1    |  ⟵————— Chip Dir.

  1               2          | page 2        page 3    |

  2               4          | page 4        page 5    |

  .               .                   .                    ⟵————— Bookkeeping
  .               .                   .                            Blocks
  .               .                   .

  19              38         | page 38       page 39   |

  20              40         | page 40       page 41   |

  21              42         | page 42       page 43   |

  22              44         | page 44       page 45   |

  .               .                   .                    ⟵————— 1003 data
  .               .                   .                            blocks
  .               .                   .

 1022            2044        | page 2044     page 2045 |

 1023            2046        | page 2046     page 2047 |
```
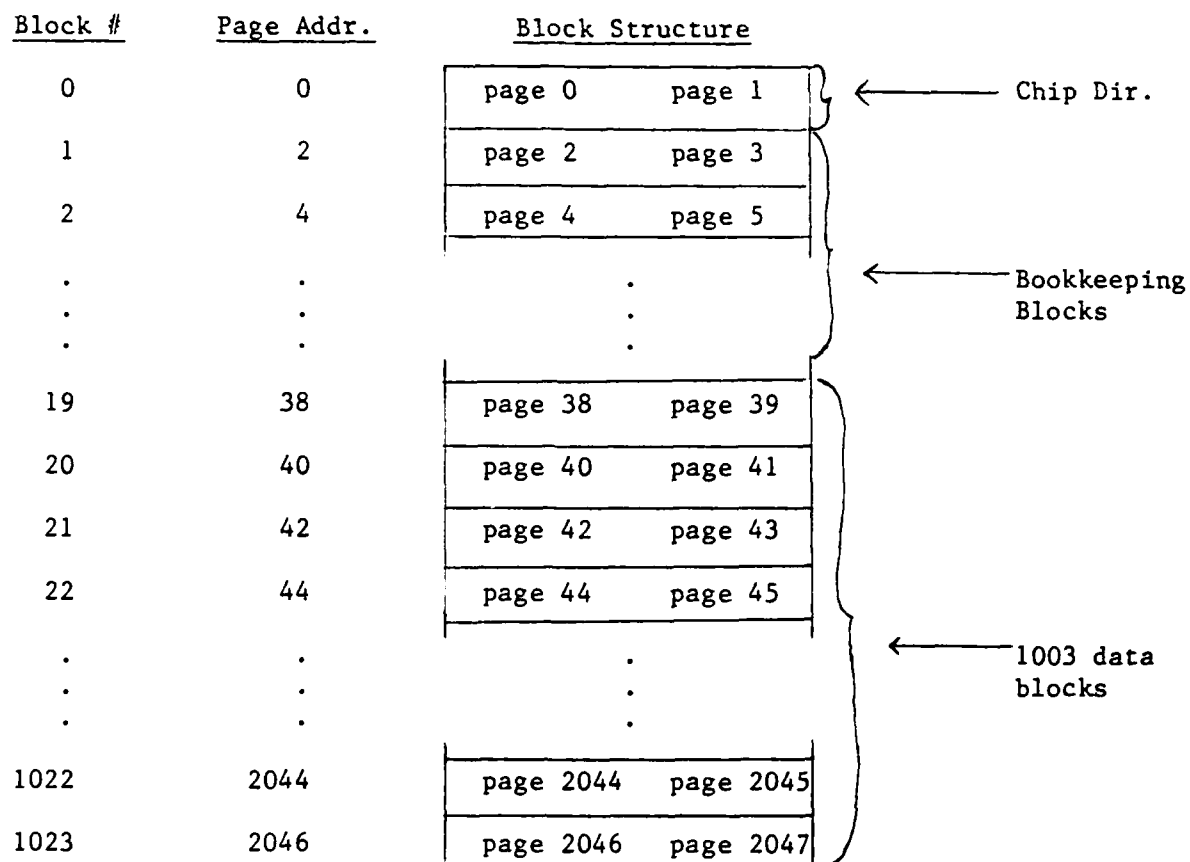
FIGURE 48: Page and Block Format of Bubble Device

```
    1      2      3       4        5        6        7        8

  | F  |  R  |  E  | Blocks | Used | Free Address | # Files |
  |    |     |     |  LSB   | MSB  |  LSB    MSB   |         |
```

FIGURE 49: First 8 Bytes in Chip Directory (block 0)

110

bytes 1,2,3

These bytes contain the characters "FRE". This field is used by the 6100 bubble software to detect a Chip Directory in the bubble at start up.

bytes 4,5

These bytes contain the LSB and MSB respectively for the number of data blocks (as opposed to bookkeeping blocks) currently in use in the bubble device. The range is 0 to 1000 inclusive.

bytes 6,7

These bytes contain the LSB and MSB respectively of the page address of the next free (unused) data block in the bubble device. Range is even numbers in the range 42 through 2046.

byte 8

This byte holds the number of current files in bubble device.

Following the usage information in the Chip Directory is room for up to 20 File Header Pointers. Each File Header Pointer is 6 bytes long and contains information pertaining to one file. This allows up to 20 files per bubble device, each file having a File Header Pointer. Each File Header Pointer has the following format (Figure 50).

1	2	3	4	5	6
char1	char2	char3	Type	Header Block Addr LSB	Blocks Used

FIGURE 50: Typical File Header Pointer in Chip Director, one of up to 20 such structures.

bytes 1,2,3

These bytes contain the characters that specify the file's name. The characters are compressed into the MACREL assembler convention (six bits per character). Due to the manner in which the 6100 software decodes these characters for printing, the first, third, and fifth characters will always print in the range A through C. The other 3 characters will print normally.

byte 4

This byte contains a number signifying the file type.

byte 5

This byte contains the LSB of a page address that points to
this file's File Header Block. File Header Blocks reside in
blocks 1 through 20, therefore the MSBs are always 0 and storage
is unnecessary. The range is 2 through 40 for the LSB.

byte 6

This byte contains the number of data blocks used by this
file. The range is 0 through 50.

Blocks 1 through 20 contain File Header Blocks, one per
file. Each one of these blocks is pointed to by the file's File
Header Pointer within the Chip Directory. The 6100 keeps a copy
of only one File Header Block in memory at a time. The 6100 copy
of this structure is called an Open File Header Block. It is the
presence of one of these Open File Header Blocks that defines a
file as "open" and therefore readable and writeable. A File
Header Block has the following format (Figure 51).

1	2	3	4	5	6		7 to 128			
char1	char2	char3	Type	Blks Used	Max Blks	LSB	Map MSB	LSB	MSB	LSB	MSB
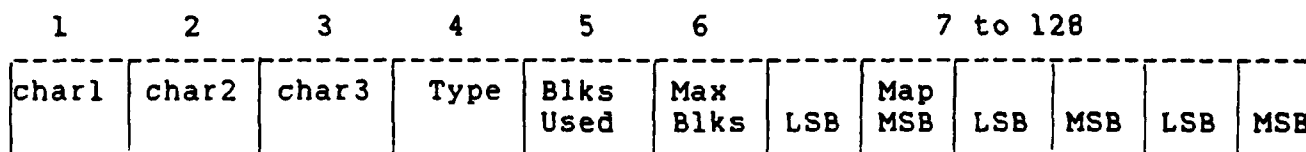
FIGURE 51: Typical File Header Block, one of 20 such
structures (blocks 1 to 20).

bytes 1,2,3

These 3 bytes contain the file's name. This field is a copy
of the name field in the file's File Header Pointer section of
the Chip Directory.

byte 4

This byte holds the file's type signifier. It is a copy of
byte 4 in the file's File Header Pointer.

byte 5

This byte holds the number of data blocks this files owns.
It is a copy of byte 6 in the file's File Header Pointer.

byte 6

This byte holds the maximum allowable number of blocks that
the file may own. It is currently set at 50.

112

bytes 7 to 128

These bytes contain a map of block addresses (2 bytes per address, LSB MSB). These addresses point to used data blocks belonging to this file. This addressing scheme actually can address more blocks than are in a bubble device, therefore the map is limited by software to be 50 addresses (100 bytes). The last 22 bytes of the map are not used.

The remainder of the bubble (blocks 21 to 1023) are used to store the actual file data. A complete file structure map is shown in Figure 52.

The remaining routines of the FSH are essentially logic and error testing wrapped around calls to the reader/writer primitive.

The present filing strategy is rather simple. To create a new file, the host issues the create command and supplies a three character name and one character type for that file. The bubble computer will then create the file with the specified name, returning an integer as that file's number. Thereafter, all communication pertaining to that file require the use of the file's number. Once the file has been created, it may be written to or read from. Whether or not the file is actually open for such operations is determined by the presence or not of an Open File Header Block in the bubble computer memory. The host requests for a read or write need not be concerned with whether the target file is open or not. If the bubble computer determines that the target file is not open, then the current file is closed and the proper file is opened prior to the actual read or write. The filing system is therefore able to support random access to any one data block (during a read), although the current host software does not require this feature. Presently all data is to be stored sequentially, filling the files up one after another. There are no host data retrieval requirements during a mission to date.

11.3 Host Computer Bubble Memory Task

Bubble memory software on the host computer side of the communications link is far less complex than the bubble computer operating system. In the current implementation, this software is a separate task running under the vehicle command computer operating system (VOS). Running concurrently with the bubble task is the vehicle mission task. The mission task generates data to be stored in small chunks (about 40 bytes) and sends them to the bubble task using VOS's intertask message mechanism. The bubble task collects these chunks into block sized entities (128 bytes) before attempting to actually store them in the bubble device. To store the collected block, a packet is built around it, and then the packet is sent down the serial link to the bubble computer. The bubble computer will answer the store request according to the success or failure of the transmission. Built in to the

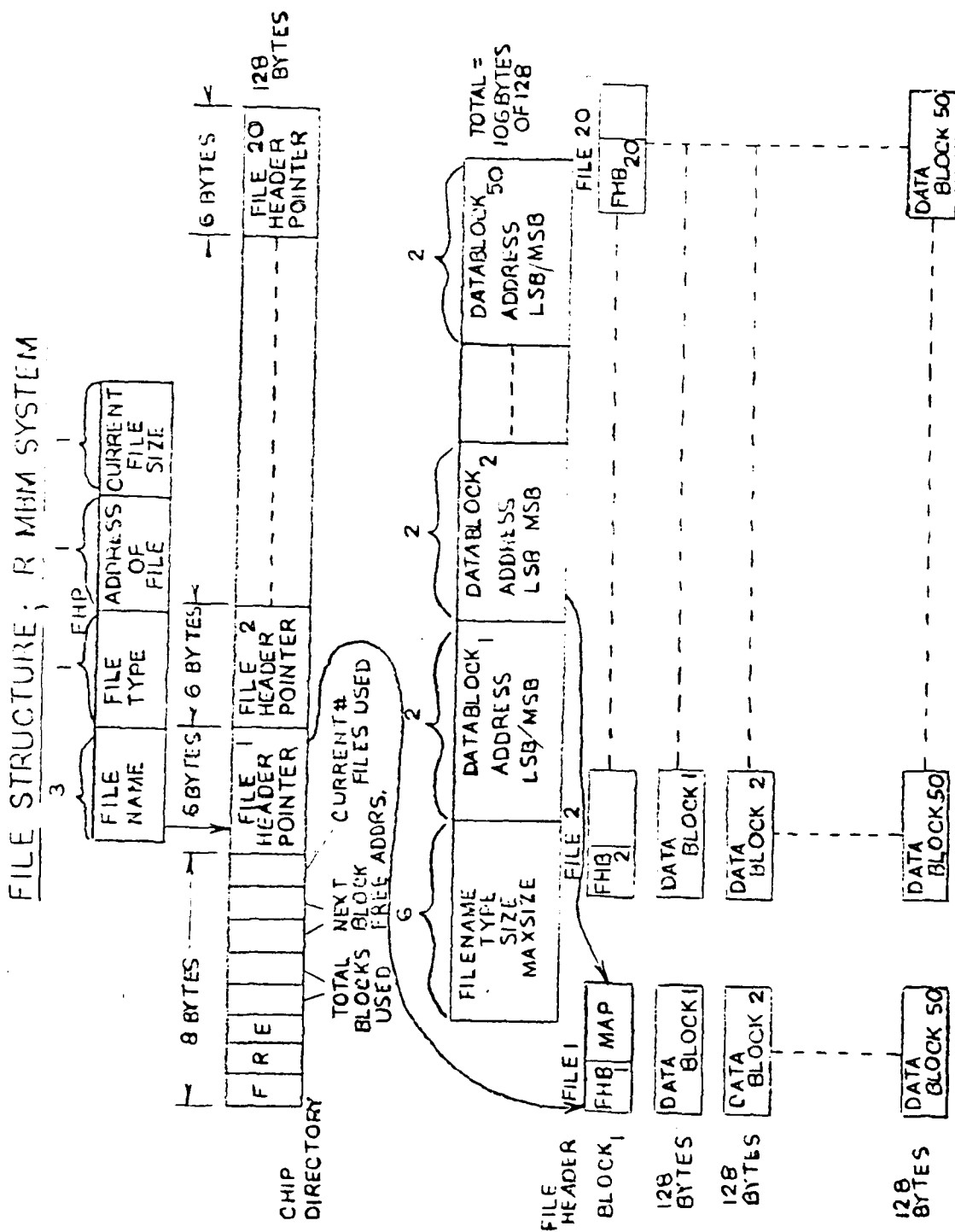FILE STRUCTURE; R MEM SYSTEM

FIGURE 52

bubble task is the intelligence to handle checksum errors and retransmission requests.

The bubble task was designed to completely handle communications with the bubble computer. The mission task need only send its data packets to the bubble task and forget about them. The present mission task does not require past data to be retrieved from the bubble, but the software mechanisms necessary to implement such activity exists.

11.4 MBM Recorder

Figure 53 is a sample of a printout of the MBM data recorder. Each 4 line segment is a record of vehicle performance data which is updated and recorded at every command computer cycle (~1.5 seconds). Figure 54 provides a detailed explanation of the data printout format.

11.5 MBM Plotting Routine

In the course of a mission the Magnetic Bubble Memory stores vital information about the vehicle's operation. This information must be extracted and analyzed after each day of testing to evaluate vehicle performance and make modifications. Since each day's testing can fill all 128 Kbytes of the bubble memory the task of sorting and analyzing the data becomes tedious and at times overwhelming. In the present situation, it is desirable to interpret the data, make software corrections, and continue testing within a day. The most meaningful way to represent the position and bearing data is through plotting, but hard plotting of the 12,000 points is very tedious and would require over 20 hours.

In order to speed the data interpretation an automated plotting package, plotrec.c, has been developed. This software runs under the standard Vehicle Operating Software (VOS), used on the EAVE vehicle. With VOS and the plotting package running on the 68000, the 6100 bubble computer is interogated and data is transferred, one record at a time. The program then drives an Anadex 9501 alphanumeric/graphics printer to produce the plots. The printer has a density of 600 points/line which gives plot resolutions of 0.1 feet in x, y, and z, and 1 degree in bearing.

The program allows the operator to select files to be plotted and relieves the operator of further intervention. For ease of interpretation x, y, z and bearing data are plotted on the same set of axes. Also, an entire file (150 records) is plotted on one 8 1/2 x 11 inch page. In addition to the position and bearing information the vehicle command changes are noted on the plots. The time required for plotting the entire contents of the MBM is 1 1/2 hours; this time is currently limited by the response of the printer.

Figure 11 in Section 5 is an example of a mission plot made by this technique.

SAMPLE PRINTOUT FROM MBM DATA RECORDER

```
Record Time:   54:45.00    Command # 2 File # 2      Block # 9     Rec # 3
8968396 113 241 98 289 1011
 0      1272      627      521      1265     650     0     1246     633
16      16       11       10    2    -3


Record Time:   54:46.68    Command # 2 File # 2      Block # 10    Rec # 1
8968554 115 237 99  296 1011
 0      1267      635      511      1255     655     0     1244     637
13      13       11       10    0     0


Record Time:   54:48.19    Command # 2 File # 2      Block # 10    Rec # 2
8968713 118 227 99 300 1011
 0      1257      647      502      1246     670     0     1234     651
15      15        0        0    0     0


Record Time:   54:49.76    Command # 2 File # 2      Block # 10    Rec # 3
8968865 123 224 202 299 1011
 0      1250      646      473      1240     665     0     1225     647
10      10        2        2    5    -7


Record Time:   54:51.34    Command # 2 File # 2      Block # 11    Rec # 1
8969073 123 217 101 293 1011
 0      1242      654      466      1231     676     0     1217     655
11      11       -1       -1    0     1


Record Time:   54:52.93    Command # 2 File # 2      Block # 11    Rec # 2
8969182 126 211 101 287 1011
 0      1232      656      462      1255     700     0     1211     661
15      15       -4       -4    2    -3
```

FIGURE 53

116

Record Time: Min: Sec: Hundreds		Command No.	File No.	Block No.	Record No.

Time in Hundreds of a Second	X Position in tenths of a foot	Y Position in tenths of a foot	Z Position in tenths of a foot	Relative Bearing in Degrees	Reliability Word

XXXX	XXXX	XXXX	XXXX	XXXX	XXXX

↑Raw Data Matrix of 9 counts returned in navigation computer

Motor 1 Speed and Polarity	Motor 2 Speed and Polarity	Motor 3 Speed and Polarity	Motor 4 Speed and Polarity	Motor 5 Speed and Polarity	Motor 6 Speed and Polarity

↑Thruster Motor Commands sent by command computer to thruster computer

MBM DATA RECORDER PRINTOUT FORMAT

FIGURE 54

117

12. THRUSTER COMPUTER SYSTEM

The EAVE vehicle is propelled by the judicious selection of speed and polarity of six thruster motors. Figure 55 is a block diagram of the thruster computer system.

The thruster computer performs the following functions:

1. Turns thruster motors on and off.

2. Provides for positive or negative thrust.

3. Provides for any of 31 speed settings in either direction.

Each thruster can be individually addressed, hence any number of thrusters may be on and at different speeds at any one time. The decisions regarding thruster control, however, are made in the 68000 command computer. The command computer is constantly updating the thruster computer with specified parameters pertaining to each thruster. (See Section 13)

The thruster computer system is a single field 6100 CPU system. The computer operating system resides in the 3K PROM and contains all software necessary to manipulate the six thruster drivers which in turn control the thruster motor speeds and polarities. The thruster computer communicates with the command computer through the host UART and may be accessed by a terminal via the supervisor UART for manual testing and debugging.

The CPU must be provided with the address of the thruster, the direction of rotation for the thruster (polarity), and the speed desired. This information is provided by the command computer and has the following format.

Bit #   0   1   2   3   4   5   6   7   8   9   10   11

X	X	X	A	A	A	D	5	5	5	5	5

Don't Care   Thruster ID | Thruster Speed
                        Direction

Thruster speed = 5 = $[0-37]_8$ = $[0-31]_{10}$

Thruster address = A = $[1-6]_8$

Thruster direction = D = [1 = forward, 0 = back]

Don't care = X

where forward is defined as a positive thrust along the body of each motor in the direction shown below.

THRUSTER COMPUTER SYSTEM BLOCK DIAGRAM

FIGURE 55

119
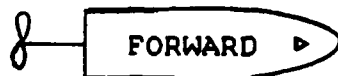
The interface card operates basically as follows (Figure 56 drawing N0100051). A 6101 PIE element handles the signal control and timing to and from the 6100 CPU. A 6440 latched decoder driver decodes the address received on the Dx lines 3, 4 and 5 and asserts the correct output line to one of six thruster driver cards when enabled. The thruster direction and speed information (Dx lines 6 through 11) is fed to a 6495 tristate buffer driver. The corresponding outputs are also sent to the thruster driver card.

A 4047 multivibrator is used to generate the system clock at 19.2 kHz. A 40103 down counter outputs a pulse every 32 clock cycles establishing the pulse width.

There are six thruster controller cards (Figure 57 drawing #100050 Rev. A), one to drive each of the six thruster motors. The sign and speed data is latched into a 40174 D type flip-flop. The sign is used to control a 28 volt 12A Potter and Brumfield relay through a 4N30 opto-isolator and a 2N3725 transistor. A 4015 shift register is used to ensure that each thruster motor responds to commands at least one clock cycle later than the others. This prevents large voltage spikes in the system. The "set" pulse sent from the 40103 on the interface card causes the speed data to be jam loaded into another 40103 down counter. Once the counter has counted the inputted number of clock cycles, it outputs a pulse to reset an RS flip-flop, and turns a 4N35 'on' which turns on two 2N5302 power transistors connected in a Darlington configuration. This allows current to flow through the motor at 24 volts. The reset pulse timing is what is controlled by the speed data. This timing allows the motor to be 'on' for varying proportions of the motor dutycycle and is effectively a pulse width modulation of the motors.

120

NOTE: R3-R9 NORMALLY NOT REQUIRED

RIBBON CONNECTOR
THRUSTER SELECT
SPEED

19,20 Vcc
9
11
13
17
18
6 LSB
14
16 MSB
12 MSB
10 96N
5 CLOCK (1200HZ)
4 SET (600HZ)
1,2 GRD

U5 4010 3
3,8,11/2,3
15
1 2,4,5,6,7,9,10,16
+V

U4 4047
4,5,6,14
13
1 2 3 7,8,9,12
R2 10K
C3 .003

U2 6495
1,15,8
16
3 5 7 9 11 13
2 4 6 10 12 14
10

R6
R R R R R 3
5 8 7 9 4

THIS IS +V
Vcc
R1 27
C2 68μ
C1 .01

U3 6440
1,18,6
2,9
7,8
3
4
5
11 12 13 14 15 6

U1 6101
1 36
2 3 13
15 16 17 18 19 20 21 22 23 24 25 26 10 28 33 34 39 40
27 14 12 8 11 9 7 6 5 4
R17 1M
R16 +V
R15
R14

SYSTEM BUS
GND 1
DX0 2
DX1 3
DX2 4
DX3 5
DX4 6
DX5 7
DX6 8
DX7 9
DX8 10
DX9 11
DX10 12
DX11 13
Vcc 14
PRIN 15
XTC 16
blank 17
INTGNT 18
blank 19
blank 20
C1 21
C2 22
MEMSTR 23
DEVSEL 24
SEL 25
blank 26
RUN 27
don't use 28
PROUT S
LXMAR T
DEVSEL U
EXTCLK V
SKPINT W
blank X
WAIT Y
blank Z
RESET AA
blank BB
XTA CC
blank DD
MEMSEL EE
x use FF

121

EAVE THRUSTER CONTROL

3 SPEED RESOLUTION

OF 6 SWITCHING BOARDS

NOTES:
ALL RESISTORS 1/4 WATT UNLESS NOTED
USE MOLEX 1490 CONNECTOR FOR +24, 24,
+ MOTOR LOADS

13. THRUSTER SUBSYSTEM SOFTWARE

The thruster motors may be addressed four different ways as follows:

1. Each thruster may be individually addressed by sending an eight bit word to the thruster command scanner, using scheme shown in Section 12, for the software control word.

2. Four parts of thrusters may be addressed together to effect different vehicle motions:

    1. Up-down thruster pair.
    2. Slide left-right thruster pair.
    3. Forward and back thruster pair.
    4. Rotate clockwise-counter clockwise pair.

    The first seven speeds are acquired by a table look up and are user selectable. Speeds 8 - 32 are used directly.

    In addition, the rotate left and right speeds may be toggled on and off with a single command to allow rapid user control over the very sensitive response of the vehicle in the horizontal plane.

3. An auto test program is included that will cycle through all six motors for all 32 speeds forward and back for hardware testing.

4. An auto altitude program may be selected that will acquire and hold a depth selected by the user.

5. A list of thruster commands may be found in Table 6.

TABLE 6

THRUSTER COMMANDS

```
/      This is the command scanner for the control
/      computer.  It will be vectored to upon input
/      or output, will decode the command and jump
/      to the proper routine.



/      The commands are:

/N.    __Set current field to N
/Carriage Return -- Close current location
/Slash -- Open current location and type contents
/Line Feed -- Close location and open the following location
/NNNNG -- transfer program control to location specified
/      BY NNNN.


/NNU -- Set up thrusters to speed NN
/NND -- Set down thrusters to speed NN
/NNF -- Set forward thrusters to speed NN
/NNB -- Set back thrusters to speed NN
/NNL -- Rotate left at speed NN
/NNR -- Rotate right at speed NN
/NNS -- Slide to starboard at speed NN
/NNP -- Slide to port at speed NN
/H ---- Halt all thrusters
/NNNNA- Set altitude to NNNN
/NT --- Auto-test thrusters [1] = on
/NNN+ - 0-6[thruster] 00-37[speed] +[forward]
/NNN- - 0-6[thruster] 00-37[speed] -[back]
/NK --- Kill printout [0] = off   [1] = on
/I ---- Initialize for interrupts
/O ---- Jump to ODT
/Z ---- Input Z coordinate
/C ---- Output thruster speeds
/NNNNX -- Set cycle rate (tdelay)
```

14.   BATTERY SYSTEM

14.1 Description (Figure 58)

The battery systems which provide power for the SIMS vehicle are comprised of two types of sealed lead acid cells.   The +8, and +16 volt systems are configured of series-parallel combinations of 'D' size 2.0VDC 2.5Ah Gates cells, (P/N 0810-004), arranged to provide the proper terminal voltage/amp hour rating. The +24V system is configured of (4) four, series connected, 6VDC 33Ah Eagle Pitcher 'Carefree Magnum' batteries (P/N CFM 6V33). The batteries are mounted in two separate, indentical cylinders, and connect to the appropriate systems via waterproof connectors.

+8VDC System

The +8VDC system consists of 16 Gates cells arranged in four banks of four cells each.   This arrangement provides 8VDC @ 10Ah per cylinder.

+16VDC System

The +16 VDC system consists of 16 Gates cells arranged in two banks of eight cells each.   This arrangement provides 16VDC @ 5Ah per cylinder.

+24VDC System

The +24VDC system consists of four Eagle Pitcher batteries in series, providing 24VDC @ 33Ah per cylinder.

Power Availability:

Power Cylinder (Stack)	Per System
+8VDC @ 10Ah	+8VDC @ 20Ah
+16VDC @ 5Ah	+16VDC @ 10Ah
+24VDC @ 33Ah	+24VDC @ 66Ah

14.2 Charging Method (Figures 59 & 60)

Each bank in the +8 and +16V systems pins out separately and may be charged separately. This method allows for differences in charging rates and times to suit each bank.   The batteries are first charged at a constant current until a pre-set voltage level is reached, which indicates state of charge, and then automatically switched to a 'tickle' or constant voltage mode and allowed to rise to their finished or fully charged potential.   A description of this method follows.

FIGURE 58 - Battery System
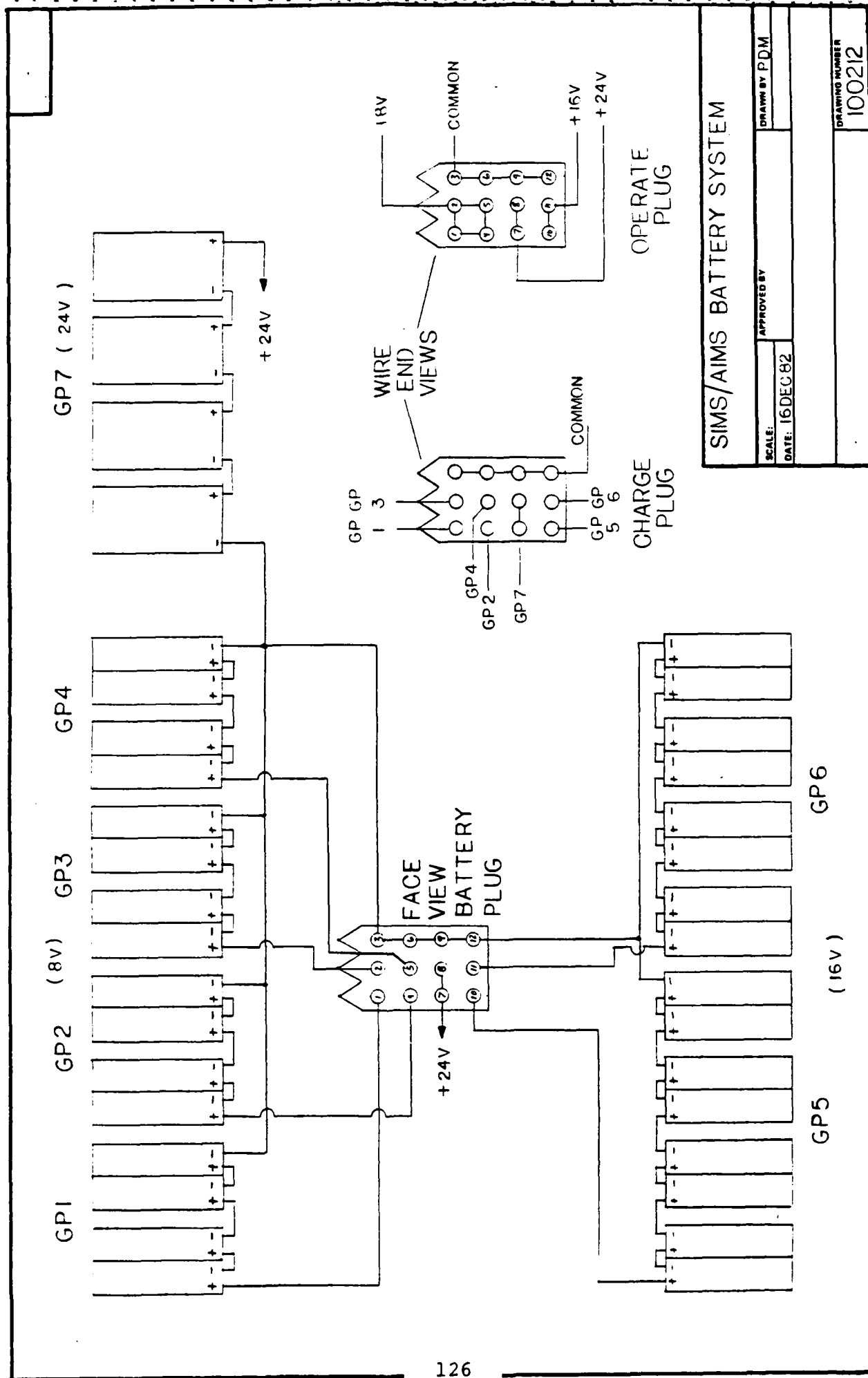
SIMS/AIMS BATTERY SYSTEM

SCALE:
DATE: 16DEC82
DRAWN BY PDM
APPROVED BY
DRAWING NUMBER 100212

126
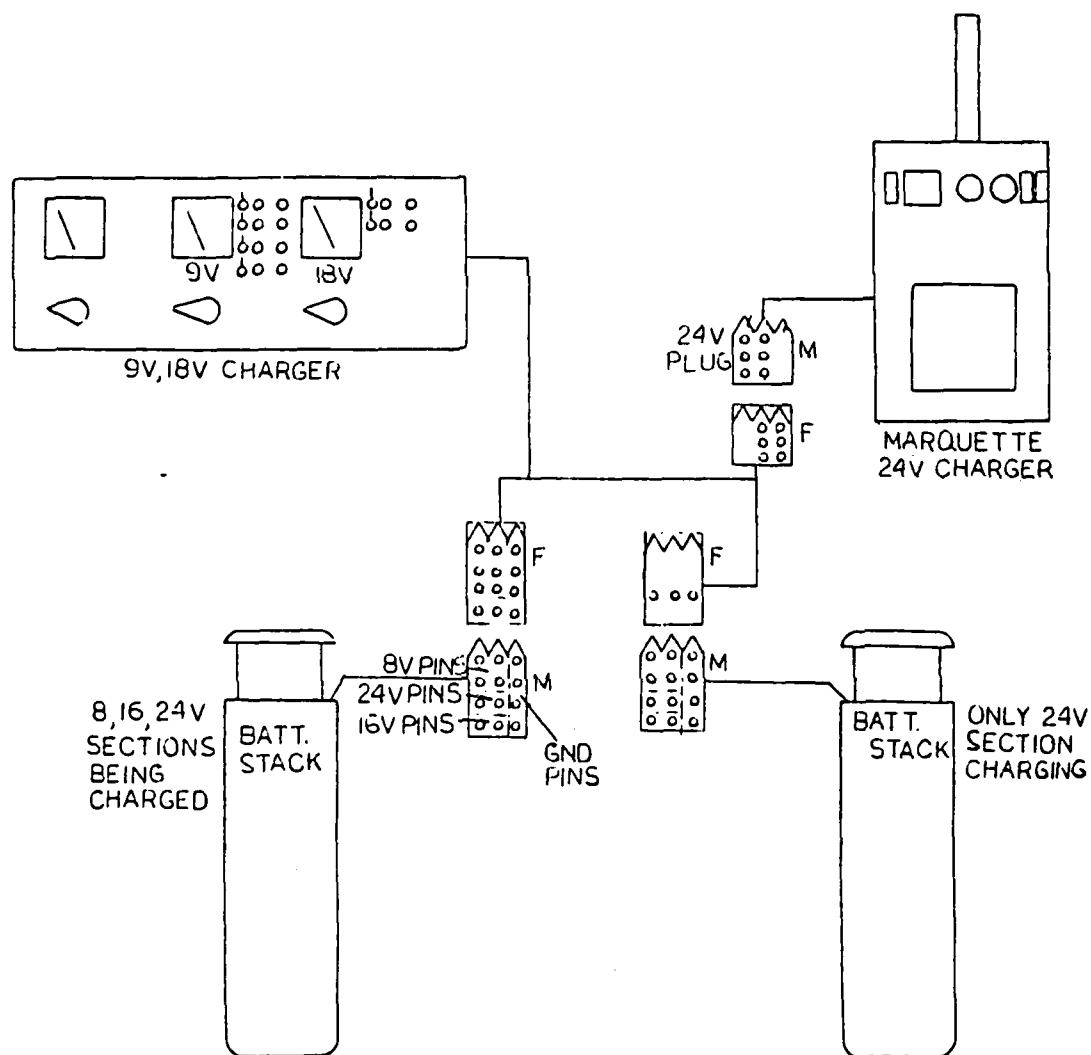
FIGURE 59 - EAVE Battery Charging Hookups
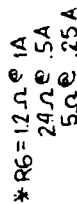
EAVE BATTERY CHARGING CIRCUIT

FIGURE 60

Each bank of the +8 and +16 VDC systems is connected to a single charger/regulator card in the battery charger (Figure 61). At turn on, the regulator, IC1 is connected through the NC contact of RY1 to R6, this establishes the mode of operation as the constant current method. As the battery bank begins to charge the voltage across the divider R8-R9 increases to a point representing approximately 90% of the full terminal voltage at which time the portion appearing across R9 trips the voltage comparator section of IC2 to switch modes. When the output of IC2 goes high it switches RY1 to the NO position changing the mode to constant voltage by inserting R5. At this time IC1 is operating at some point determined by adjustment of R4. Simultaneously the output of IC2 latches its reference input to a near ground potential to prevent returning to the constant current mode, and causes the LED indicators to indicate the proper ode of operation. The charger/regulator stays in this state of operation until turned off at the end of the charging cycle. These actions are independent of the charging modes of the other charger regulator cards in the overall system.

The +24V system is charged manually, that is, it is monitored and the charging rates and voltages varied by the attendant. The initial voltage impressed on the stack or stacks is 29VDC. At this voltage the maximum current per stack is 10 amps. The voltage is periodically checked and adjusted. When the charging current drops to 1.5 amps per stack the voltage is adjusted to 29.4 VDC and left there until the charging current drops to .2 amp or less per stack. At this current the battery stack(s) are fully charged. This process varies in time depending on the state of the battery stacks. A Marquette Charger, Model 32-175 is used in this procedure.

The battery systems as described seem to be adequate for typical SIMS missions and testing, when fully charged.

15. SOFTWARE SYSTEMS

15.1 Overview

The software system developed at MSEL for the SIMS mission is divided into three basic subsystems corresponding to the two IM6100 CPU and one M68000 CPU hardware configurations. Each of the three will be covered in detail below.

15.2 Software Development

Software for each of the two IM6100 based computers is written entirely in assembly language. Two different development systems are used in this process.

First, the University of N.H. DEC system-10 has a resident PDP-8 assembler which may be used to produce loadable object code from IM6100 assembly language files. The resulting object module may be down loaded to an intermediary medium such as magnetic tape by means of a Fortran program developed at MSEL. The stored code may then be loaded into the selected microcomputer by a ROM resident loader.

Second, an in-house DEC station-78, a PDP-8 based development system, has both a Pal-8 and Macrel-8 assembler capable of producing loadable object code from IM6100 assembly language files. These files may be down loaded to an intermediary medium or directly to the desired IM6100 based microcomputer.

Software development for the Motorola 68000 based microcomputers follows a different pattern. An in-house Emperical Research Group M68000 based development system running "Idris", a Unix like operating system by Whitesmiths Inc., is used to produce all loadable object code. Several high level languages are provided under Idris including "C" and Pascal, in addition to an assembler and linking loader. The "C" language is used extensively at MSEL for systems programming needs and for mission software development. An Idris "pre-processor" translates "C" source code into appropriate 68000 assembly language which may be examined and modified if needed before being assembled into a re-locateable object module. A number of these modules may then be linked into loadable form and saved on an intermediary medium or down loaded directly into a 68000 based microcomputer.

15.3 6100 CPU Software Monitors

All 6100 CPU based systems are controlled externally by commands issued to a resident monitor. The monitor decodes input on an interrupt driven basis and performs certain functions based on that input.

The navigation CPU, thruster CPU, and MBM CPU monitors have two entry points, one associated with the tether-user port and

one associated with the command CPU port. An interrupt generated from one of the two UARTS will cause a vector to an appropriate interrupt service routine where command decoding may take place. A flow chart of the decoding process may be found in Figure 61. Since the two ports use separate interrupt service routines, I/O at both ports may be serviced asynchronously without deadlock. At the end of the decoding process a global flag is set in memory to indicate a desired function is to be activated. All of the above activity occurs within the interrupt environment of the monitor.

All vehicle functions, however, are performed outside of the interrupt structure in a circular task routine. After initialization, control passes into an infinite loop of short routines, each of which checks for a flag set by the decoder for a particular function. If the flag is set, the function is performed, if not, the function is skipped and the next flag is checked. Control spins forever in this interruptable loop allowing both the command CPU and the user to exercise their options asynchronously. A list of available commands for the various 6100 CPU's may be found in Figure 62.

15.4 User Interface (U/I)

In order to provide maximum user control over all vehicle systems, both hardware and software, a user interface was created. The U/I is itself a task running within the O/S multitasking environment in parallel with the monitor task and others that may exist. It acts as a shell surrounding all other vehicle software and provides the user with a variety of tools to shape, test, and run different mission scenarios both in-house and in the field.

The U/I is constructed as a hierarchical tree structure with, at present, three levels of user interaction (see Figure 63). On start-up the user is prompted to select a function from a menu of available alternatives. With each selection the user descends to a deeper level and is prompted once again until he reaches the level at which a desired function resides. He is then prompted to make changes in parameters, exercise hardware, run mission software, etc. until he is finished with the particular function. He may then ascend the tree and descend at will choosing the various procedures necessary to accomplish a particular mission.

There are a number of advantages to a tree structured, menu driven U/I.

1. User friendliness - The user is fully prompted at all times and is thus immediately made aware of what powers are available to him and how to exercise them properly. The system is protected against user error as incorrect and potentially damaging input is screened and the user notified of his mistakes.
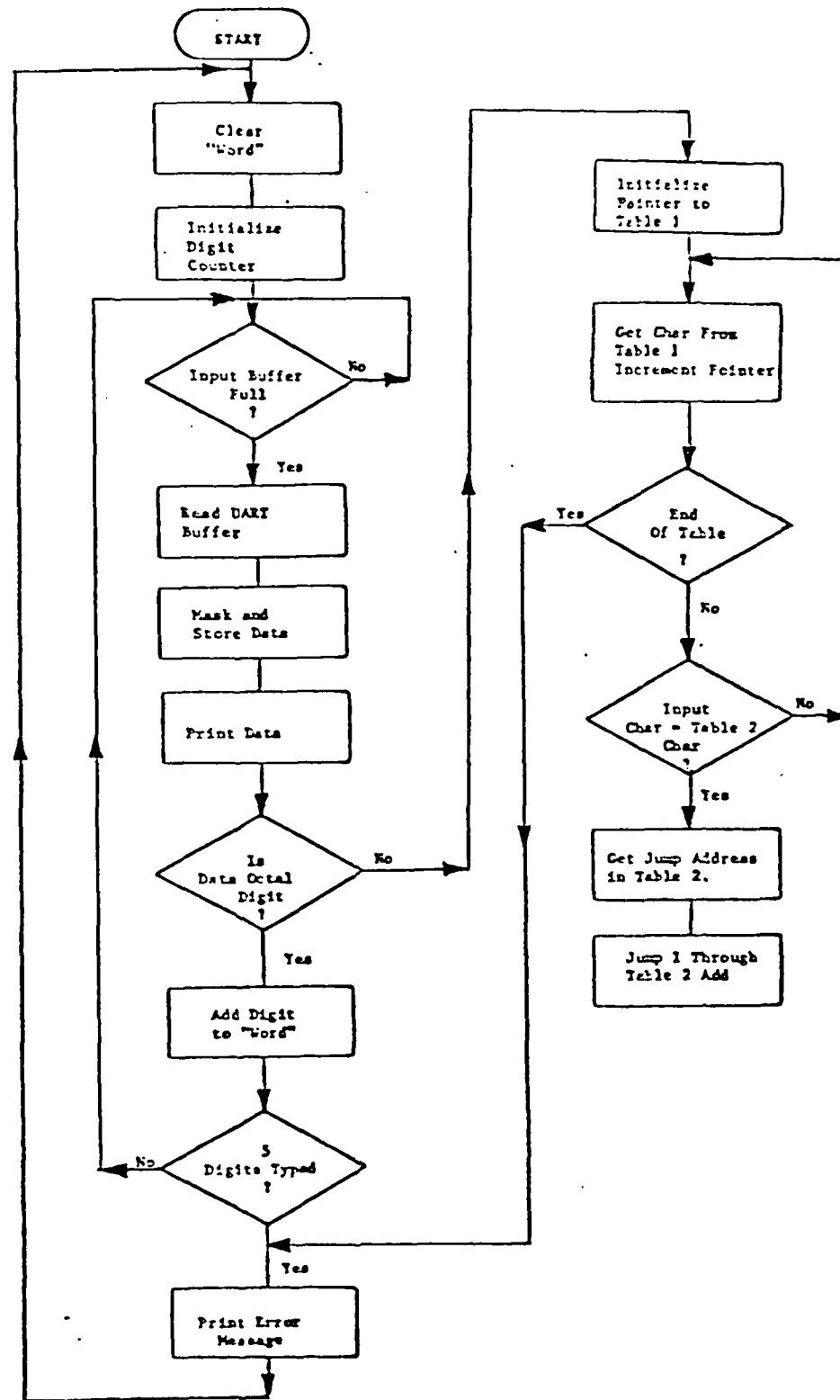
131

COMMAND SCANNER

FIGURE 61

```
01800
01900     /         THIS IS THE COMMAND SCANNER FOR THE CONTROL
02000
02100     /         COMPUTER. IT WILL BE VECTORED TO UPON INPUT
02200
02300     /         OR OUTPUT, WILL DECODE THE COMMAND AND JUM
02400
02500     /         TO THE PROPER ROUTINE.
02600
02700
02800
02900     /THE COMMANDS ARE:
03000
03100
03200
03300     /N.    __  SET CURRENT FIELD TO N
03400
03500     /CARRIAGE RETURN -- CLOSE CURRENT LOCATION
03600
03700     /SLASH -- OPEN CURRENT LOCATION AND TYPE CONTENTS
03800
03900     /LINE FEED -- CLOSE LOCATION AND OPEN THE FOLLOWIN
04000
04100     /NNNNG -- TRANSFER PROGRAM CONTROL TO LOCATION SPE
04200
04300     /          BY NNNN.
04400
04500     /C [LEAR BUFFER] (400 - 3777) OCTAL
04600
04700     /D [UMP TO TAPE]
04800
04900     /L [OAD TO MEMORY]
05000
05100     /R [ECORD #]
05200
05300     /S [PEED OF O/S] (SEE REWAIT) (INIT TO 7777)
```

```
05400
05500     /P [ING TRANSPONDER] <0> = OFF, <1> = ON
05600
05700     /H [CLOCKF TIME] (INITIALLY 7000)
05800
05900     /N [UMBER OF RECORDS]
06000
06100     /K [ILL PRINTOUT] <0> = OFF, <1> = ON
06200
06300     /E [CHO KEYBOARD] <0> = OFF, <1> = ON
06400
06500     /I [NITIALIZE O/S]
06600
06700     /A [CTIVATE NAV SYSTEM] <0> = ON, <1> = OFF
06800
06900     /Z [COORDINATE INPUT]
07000
07100     /E [LOCK DATA OUTPUT]
07200
07300     /M [ATH PROCESSOR OFF]
07400
07500     /X [DUCER #] <1 2 OR 3> (INITIALLY 1)
07600
07700     /O [CT]
07800
07900     /T [RANSFER ROM] ( <1> = TRFR CORBUF TO ROM )
08000
08100     /J [UMP AROUND FLD 0 OUTPUT] <0> = PRINT
```

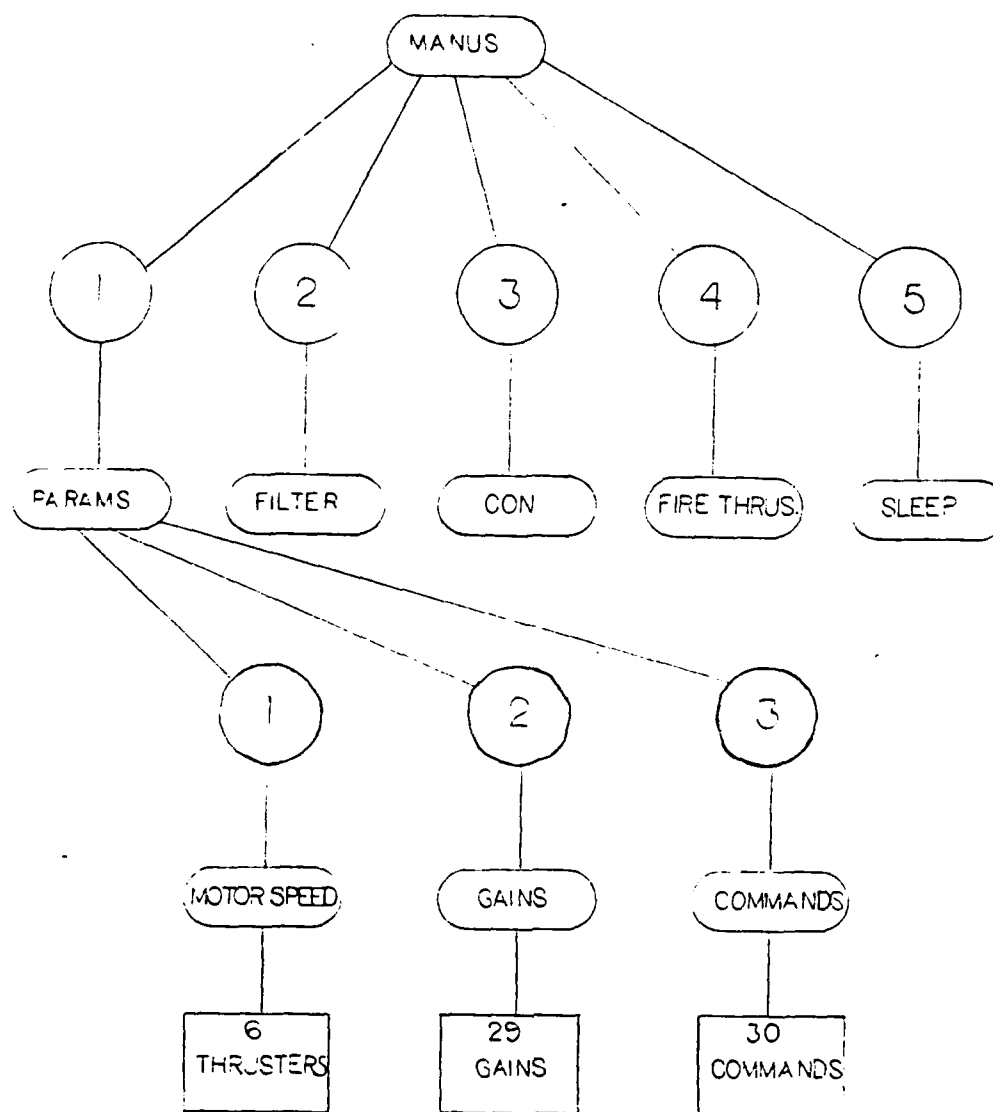FIGURE 62:   AVAILABLE 6100 CPU COMMANDS

14

FIGURE 63

134

2. Custom mission creation – with a small map of the command structure, the user may quickly perform complicated combinations of functions to create a particular mission scenario which remains in effect until changed. The user may then exercise the mission repeatedly making changes dynamically as the results are observed. This method allows great flexibility in the system testing and modification phase, particularly in the field where the facilities for software re-write/re-compilation are not available.

3. Modularity – Each node of the U/I tree is constructed as a "C" function with well defined parameters. This partitioning allows each function to be written, tested and called independently. Each function may be used by any other at any time, thus allowing a complicated interaction of functions while retaining their individual simplicity. Modularization also simplifies the task of adding to or modifying the basic structure of the U/I. As each "module" is built up out of the same basic components, they may be used as building blocks to create larger systems more easily.

An examination of the current built-in features of the U/I follows.

When the U/I is first activated it prompts the user to select a desired function from a menu displayed on the screen by typing a number from 0 to the maximum. Any input other than that requested will be rejected and the user will be informed of an error and will be re-prompted. By typing a "0" the user may exit the current routine and ascent the tree structure to the level above where he is prompted again with a new menu of available selections. If he is at the highest level a "0" input will exit the U/I and return to the O/S. An entry of "1" to the maximum available will either perform a function or descend the menu tree and prompt for another selection. A description of the available functions at each level follows:

I. Parameters (select parameter to change)

1. Motor speeds (select motors 1-6 at speeds ± 31)
2. Gains (select gains 1-27 see Figure 64)
3. Commands (select commands 1-30)
    1. Type (enter type of command)
    2. Ignorexy (ignore x, y computations)
    3. Duration (time limit of command)
    4. Position (enter desired position)
        1. x
        2. y
        3. z
        4. xyz speed (maximum)
        5. bearing
        6. bearing speed (maximum)
    5. Display current command (see Figure 65 for example)

135

```
/*----------------------------------------------------------------- */
     /* defines for CONtroller task */

     /* default gains values for the variables used in control */

#define DRTB         0.7      /*   1  position */
#define DRTC         50.0     /*   2  velocity */
#define DPTA         0.0      /*   3  integral */
#define DPTB         0.28     /*   4  position */
#define DPTC         20.0     /*   5  velocity */
#define DYTA         0.0      /*   6  integral */
#define DYTB         0.28     /*   7  position */
#define DYTC         20.0     /*   8  velocity */
#define DZTA         0.05     /*   9  integral */
#define DZTB         0.25     /*  10  position */
#define DZTC         35.0     /*  11  velocity */

#define XPROP        2.0      /*  12  limit for integral action (feet) */
#define RPMMAX       31.0     /*  13  top motor speed number */
#define NEAR         1.0      /*  14  to target (feet) */
#define CLOSE_F      0.05     /*  15  to heading (radians) */
#define SMLDIS       0.01     /*  16  a small distance (feet) */
#define FILCYCLES    10       /*  17  # filter cycles */
#define CONCYCLES    10       /*  18  # control cycles before exit */
#define BOFFSET      338      /*  19  compass offset from x axis */
#define MODELAY      4.0      /*  20  thruster delay, in hundredths of sec. */
#define MP1          10.0     /*  21  (mot1 polarity * power factor) max = 10 */
#define MP2          10.0     /*  22  (mot2 polarity * power factor) */
#define MP3          9.0      /*  23  (mot3 polarity * power factor) */
#define MP4          8.0      /*  24  (mot4 polarity * power factor) */
#define MP5          -10.0    /*  25  (mot5 polarity * power factor) */
#define MP6          10.0     /*  26  (mot5 polarity * power factor) */
#define ZDELAY       7.0      /*  27  downburst time (seconds) */
#define STDELAY      60.0     /*  28  tether pull delay (seconds) */
#define ZBIAS        6.0      /*  29  bouyancy bias */
#define PDELAY       30.0     /*  30  thrust pulse delay, hundredths of sec */
#define BMAX         31       /*  31-34 clippers */
#define XMAX         31
#define YMAX         31
#define ZMAX         31
#define TETHERON     1.0      /*  35  console output switch; ON = 1.0 */
#define ERRWIN       8.0      /*  36  error window for filter */
#define FILTRYS      1.0      /*  37  # times to try for good position */
#define INITSEQ      4.0      /*  38  # times to try for init. position */
#define BADRTNS      3.0      /*  39  # bad filter returns con tolerates */
#define NUMGAINS     40       /*  the number of gains + 1 */
#define MAXGAINS     50       /*  maximum number of gains */
#define NUMCOMS      31       /*  the number of commands + 1 */
```

FIGURE 64

136

```
/*-----------------------------------------------------------------*
/* array of NUMCOMS commands; 0th element is default emergency exit */
COMM_BLOCK   comms[NUMCOMS]{

/*  0 */ EXIT,          FALSE,  100,       1.1,   2.2,   3.3,  4.4,  5.55,  6.666

/*        type          ignore xy  duration   x      y      z    xdot  bear   bdot

/*  1 */ VERT_MOVE,     FALSE,  6000,      5.0,  40.0,  10.0,  1.0,  4.90,  0.225
/*  2 */ HORIZ_MOVE,    FALSE,  4000,     12.5,  20.6,  10.0,  1.0,  0.00,  0.225
/*  3 */ ROTATE_ONLY,   FALSE,  6000,     12.5,  20.6,  10.0,  1.0,  0.33,  0.225
/*  4 */ NOP,           TRUE,   000,       1.1,   2.2,   3.3,  4.4,  5.55,  6.666
/*  5 */ NOP,           TRUE,   000,       1.1,   2.2,   3.3,  4.4,  5.55,  6.666

/* now follow the preprogrammed commands */

/*  6 */ VERT_MOVE,     FALSE,  6000,     12.5,  20.6,  12.5,  1.0,  0.33,  0.225
/*  7 */ HORIZ_MOVE,    FALSE,  6000,     23.5,  26.0,  12.5,  1.0,  0.00,  0.225
/*  8 */ ROTATE_ONLY,   FALSE,  6000,     23.5,  26.0,  12.5,  1.0,  1.90,  0.225
/*  9 */ HORIZ_MOVE,    FALSE,  3000,     23.0,  27.0,  12.5,  1.0,  0.00,  0.225
/* 10 */ HORIZ_MOVE,    FALSE,  3000,     23.5,  26.0,  12.5,  1.0,  3.14,  0.225
/* 11 */ HORIZ_MOVE,    FALSE,  6000,     28.0,  29.5,  12.5,  1.0,  1.57,  0.225
/* 12 */ VERT_MOVE,     FALSE,  3000,     28.0,  29.5,   8.5,  1.0,  1.90,  0.225
/* 13 */ VERT_MOVE,     FALSE,  3000,     28.0,  29.5,  12.5,  1.0,  1.50,  0.225
/* 14 */ HORIZ_MOVE,    FALSE,  6000,     23.5,  26.0,  12.5,  1.0,  4.71,  0.225
/* 15 */ VERT_MOVE,     FALSE,  4000,     23.5,  26.0,   8.5,  1.0,  1.50,  0.225
/* 16 */ HORIZ_MOVE,    FALSE,  6000,     19.0,  33.5,   8.5,  1.0,  0.00,  0.225
/* 17 */ VERT_MOVE,     FALSE,  4000,     19.0,  33.5,  12.0,  1.0,  1.50,  0.225
/* 18 */ ROTATE_ONLY,   FALSE,  6000,     19.0,  33.5,  12.0,  1.0,  0.33,  0.225
/* 19 */ HORIZ_MOVE,    FALSE,  6000,     24.0,  35.5,  12.0,  1.0,  0.00,  0.225
/* 20 */ ROTATE_ONLY,   FALSE,  6000,     24.0,  35.5,  12.0,  1.0,  5.04,  0.225
/* 21 */ HORIZ_MOVE,    FALSE,  6000,     19.0,  33.5,  12.0,  1.0,  1.57,  0.225
/* 22 */ ROTATE_ONLY,   FALSE,  6000,     19.0,  33.5,  12.0,  1.0,  3.45,  0.225
/* 23 */ HORIZ_MOVE,    FALSE,  6000,      8.5,  27.7,  12.0,  1.0,  0.00,  0.225
/* 24 */ HORIZ_MOVE,    FALSE,  6000,     12.5,  20.6,  12.0,  1.0,  4.71,  0.225
/* 25 */ VERT_MOVE,     FALSE,  4000,     12.5,  20.6,  10.0,  1.0,  3.45,  0.225
/* 26 */ ROTATE_ONLY,   FALSE,  6000,     12.5,  20.6,  10.0,  1.0,  2.00,  0.225
/* 27 */ HORIZ_MOVE,    FALSE,  6000,      5.0,  40.0,  10.0,  1.0,  0.00,  0.225
/* 28 */ VERT_MOVE,     FALSE,  3000,      5.0,  40.0,   6.0,  1.0,  2.00,  0.225
/* 29 */ EXIT,          FALSE,  100,       1.1,   2.2,   3.3,  4.4,  5.55,  6.666

/* 30 */ EXIT,          FALSE,  100,       1.1,   2.2,   3.3,  4.4,  5.55,  6.666
            };
```

FIGURE 65

II.   Filter (run data acquisition software alone to test hardware)

III. Control (run full mission)

IV.   Thrusters (power thrusters as per set-up in motor speed setting routine)

V.   Sleep (put U/I to sleep for 2Ø secs to access O/S directly)

15.5 Hydrodynamic Control Software

In order to optimize control of the EAVE vehicle in the marine environment a hydrodynamic stability and control analysis was produced by Aeronautical Research Associates of Princeton*. This study produced the six-degree-of-freedom equations of motion with numerical values for all terms necessary for simulation of submerged maneuvering. From this information a control algorithm was implemented in the "C" language and, along with various other software modules makes up the control software package (CSP).

Individual modules of the CSP are examined below.

15.5.1 Control Algorithm

The control algorithm developed at MSEL for the SIMS program may be described as a "modified proportional integrator derivative controller". A block diagram of the algorithm may be found in Figure 66. The basic equation of motion for the vehicle in each degree of freedom is expressed as a second order differential equation. This, together with the integral control action makes the system third order: (dynamics of the thruster are ignored in this representation).

$$I\,\ddot{\theta}o \;+\; t\dot{\theta}_o \;-\; Kt[Ks(Kp\,\boldsymbol{\varepsilon} \;+\; KI \int^{\tau} \boldsymbol{\varepsilon}d\tau \;+\; \dot{\theta}o)] = 0$$

Where: I and t = vehicle inertial and damping components
          Ks, Kp, KI = controller gain parameters
          Kt = thruster gain (force/unit command input)
          θI = desired position
          θo = current position
          $\boldsymbol{\varepsilon}$ = θI $-\theta$o or error signal

*This work is documented in MSEL Report #82-1, "User's Guide for the UNHTRAJ Computer Program," January 1982 and MSEL Report #82-2 (ARAP Tech. Memo No. 82-2), "Hydrodynamic Stability and Control Analyses of the UNH-EAVE Autonomous Underwater Vehicle," January 1982. Both prepared by Aeronautical Research Associates of Princeton, Inc. under subcontract #82-Ø5.

138
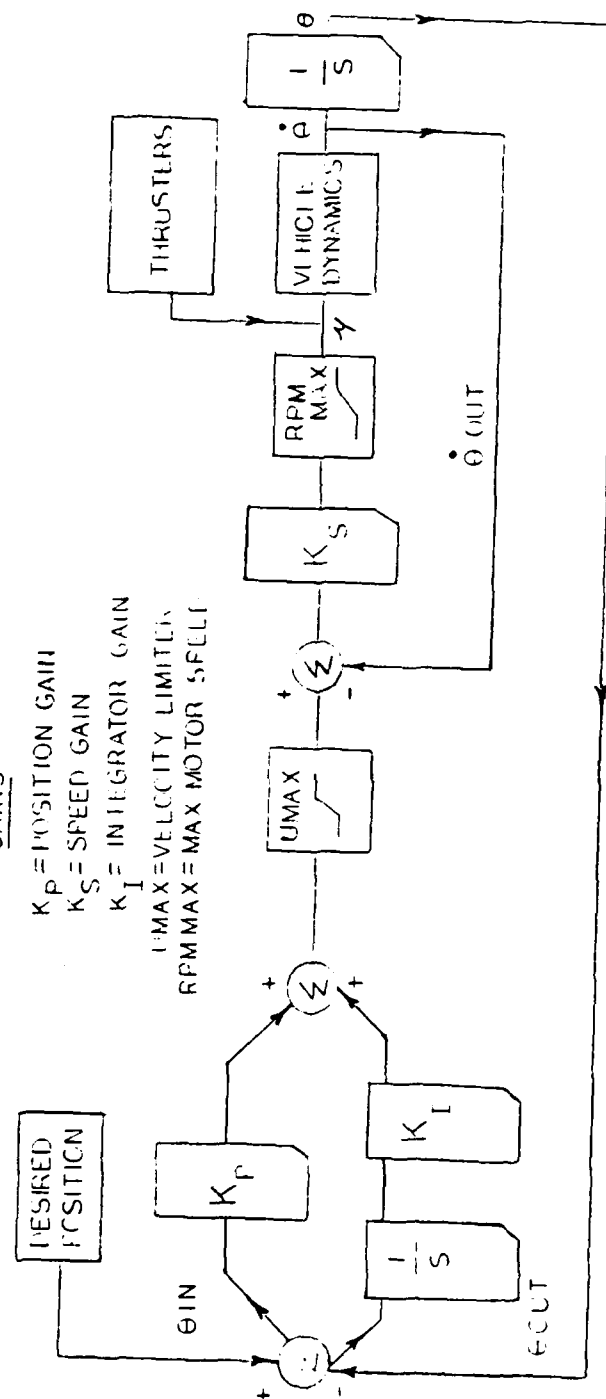
MODIFIED PROPORTIONAL INTEGRATOR DERIVATIVE CONTROLLER

MODIFIED BY: 1 VEHICLE VELOCITY LIMITER
2 THRUSTER SPEED LIMITER

FIGURE 66

139

Derivative feedback has the effect of increasing the damping of the system, thus reducing overshoot and oscillation. Proper selection of the position and velocity gain multipliers K_p, K_z should produce a critically damped system capable of reducing the error signal, $(\theta c - \theta o)$, to some value near zero thus maneuvering the vehicle in each degree of freedom to within a specified small distance of the desired position. Within this "small distance", an integral control method is added to the control system to reduce the gap between the current and desired positions still further. This is accomplished by adding to the error signal a term that is proportional to the integral over time of $\theta i - \theta o$. The addition of the integral term produces an effect such that the longer the error persists, the longer the term will become and the stronger the response will be in attempting to reduce that error. Proper selection of the integral gain multiplier should provide the last small thrust needed to acquire the final desired position. A portion of the algorithm implementing these two methods in the "C" language is found in Figure 67. The algorithm is a "modified" PID because of two additions. The summed value for the proportional (position) gain and the integrator gain is clipped by a selected maximum, "UMAX", to provide speed regulation for long traverses. Also the final value for thruster motor speed is clipped to a practical range of thruster capability.

A flow chart of the basic control algorithm may be found in Figure 68. On start-up, a command is read from a mission command list. There are six basic commands available to the user:

1. Hover - maintain the current position.

2. Horizontal move - move to position x, y.

3. Vertical move - move to depth z.

4. Rotate - rotate to bearing $\emptyset$.

5. Exit - exit the controller.

6. NOP - read next command.

Each command is an array that contains within it the basic information necessary for the control software to function. This information is as follows:

1. Duration - minimum command execution time.

2. Ignore xy - ignore current x, y in calculations.

3. x - desired x position.

4. y - desired y position.

5. z - desired z position.

140

PID CONTROLLER IN "C"

1. UCOM = (kp x dist) + (ki x integral);

2. UCOM = limit (UCOM, UMAX);

3. SPEED = ks x (UCOM - velocity)

4. SPEED = limit (speed, rpm max);

kp = position gain

ks = speed gain

ki = integral gain

UMAX = velocity limit

rmp max = thruster limit

PORTION OF CONTROL ALGORITHM/GAIN RELATIONSHIPS

FIGURE 67

FIGURE 68
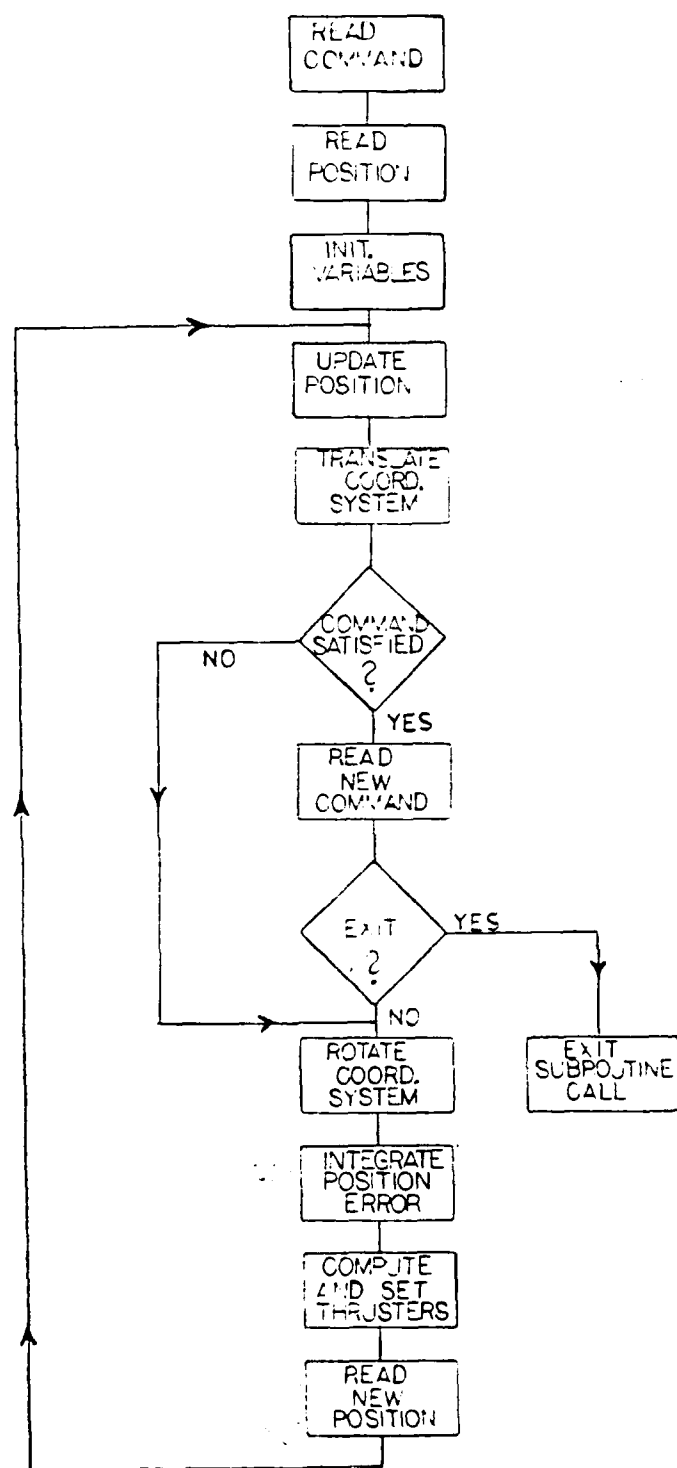
142

6. Umax - maximum velocity in x, y, z.

7. Bearing - desired bearing.

8. Rmax - maximum rotational velocity.

The command list is generated to reflect a specific mission scenario. An example of the SIMS mission scenario is found in Figure 65. After the first command is read the first position is read and all parameters are initialized. Control then falls into the main controller loop.

First the coordinate system is translated to give the present location relative to the next target point. If the command is "hover" the time is checked against the command duration. For horizontal, vertical and rotational commands the current position is checked against the desired position. If the conditions are met a new command is read, providing the duration time for that command has expired. If not, the coordinate system is rotated parallel to the desired vehicle heading so that it is along the path or perpendicular to it.

The position errors, the differences between the desired and current positions is then calculated and if this value is less than a specified amount, the integrals of these are taken. This information is then passed through the control algorithm as described above and the four thruster speeds and durations are calculated.

Because the vehicle response to rotation is far less damped than is found in horizontal or vertical motion, it is necessary to scale down the thruster speeds used for rotational maneuvers. This is accomplished by pulsing the rotational thrusters for a specified time during the control cycle. This has the effect of adding a rotational component to any other thrusts in the x, y plane. Since the pulse time is user selectable, the gain factor may be chosen to optimize stability in the x, y plane.

After the calculated thruster command has been sent to the thruster computer a new position is acquired from the navigation computer and the cycle begins again.

15.5.2 Data Acquisition

The input information needed to drive the control software in its attempt to maneuver the vehicle is produced by several data acquisition routines. The depth of the vehicle in the z direction is read from a pressure transducer while the vehicle bearing is read from a magnetic compass. Both pieces of information are adjusted for offset and scale by user selectable parameters. The position in the x, y plane is acquired on demand from the navigation computer. Time is taken from a real time clock in hundredths of a second from a pre-set start time.

143

15.5.3 Data Filter

In order for the raw data received at each data acquisition cycle to be useful to the controller, various pieces of it must be filtered in certain ways. The x, y position is calculated at the location on the vehicle of one of the three hydrophones used. This position must be translated to the geometric center of the vehicle for consistency.

The velocities in the x, y and z directions and the rate of change of the vehicle bearing are calculated. All parameters received from the navigation CPU are 12 bit words sent as two eight bit words, a high and low order 6 bit byte. These must be converted into one 16 bit word for the command CPU.

15.5.4 Thruster Interface

The outputs of the control software are four floating point parameters that correspond to thrusters 1 and 2 combined for the z direction, 5 and 6 combined for the slide motion and 3 and 4 addressed separately for the forward-back and rotate motions

These floating point values are converted by the thruster interface into four 12 bit integer parameters that are sent to the thruster CPU.

15.6 Self Calibration

15.6.1 Description

The vehicle's navigation computations require that the coordinates of the three transponders be known. A self calibration program has been developed that determines these coordinates from range data and an initial guess. The only information that the operator need supply the self calibration program is an initial guess of the x, y, z location of each of the three transponders. Presumably this will be provided at the start of a mission. The next step is to obtain a set of range data. The vehicle will execute a survey path over the transponders, collecting and storing range data to each transponder from several points along the travel path. The current simulation stores 80 sets of ranges. With each range set a depth reading is read from the pressure sensor.

The program now enters the major calculation loop. For each range set a calculated depth is subtracted from the actual depth reading produced by the pressure sensor. The differences errors from all survey points are added together to produce the total error value. If this total error is above a preset threshold, then further calculations to determine the direction of minimum error are performed, and the transponder coordinate guess is adjusted accordingly. Control returns to the top of the loop and the error calculations are performed again with the adjusted transponder coordinate guess. This transponder adjustment

continues until the total error falls below the preset threshold and the algorithm terminates with a modified set of transponder coordinates. These coordinates may then be passed on to the navigation program.

The current running version of the self calibration program is written in Pascal on the University VAX. A vehicle version of the program is being prepared in C for testing.

15.6.2 Self Calibration Program

Algorithm:

```
(* for testing, generate a set of ranges *)
get transponder coordinates
for i : = 1 to maxpoints do
      get vehicle x,y,z
      calculate ranges, add error, store
(* end data simulation *)
(* real data would be read in at this point *)
while not done do
      begin
      get initial transponder coordinate guess
      while not done do
            begin
            calculate total error
            calculate partials
            if total error < max allowable error then
                  done := true
            else
                  determine path of minimum error
                  adjust transponder coordinates in that
                     direction
                  increment iterations
                  end while loop
            end while loop
            write number of iterations
            end program
```

For simulation purposes, the first part of the program generates a set of ranges to work with. The program requires two types of input to accomplish this. The first is an x, y, z coordinate for each transponder. Second, a set of x, y, z coordinates that describe a vehicle path over the transponder array is input. As each vehicle point is entered, the ranges to the three transponders are calculated, a random noise value of 0.5% is added in, and the three results are stored. The finished product is a set of ranges with some error added in that describes a path over the transponder array. This range set is now given to the second portion of the program as if it were true field data. An actual mission will produce a similar array of range information.

To start processing the range values into transponder coordinates, the program requires an initial x, y, z guess for

each transponder. The remainder of the program is the calcula-
tion loop. Each pass through the loop modifies the transponder
coordinates in the direction that minimizes the error. The loop
is halted when the total error value is minimized below some
minimum error constant. Upon exiting the loop, the number of
iterations is printed.

15.6.3 Equations for Self Calibration Algorithm

Spheres A. $X^2 + Y^2 + (Z - C_1)^2 = R_1^2$

B. $(X - a_2)^2 + Y^2 + (Z - C_2)^2 = R_2^2$

C. $(X - a_3)^2 + (Y - b_3)^2 + (Z - C_3)^2 = R_3^2$

The intersection of A and B (B-A) yields a plane:

1. $-2a_2 X + a_2^2 + (2C_1 - 2C_2) Z + C_2^2 - C_1^2 = r_2^2 - r_1^2$

The intersection of A and C (C-A) yields a plane

2. $-2a_3 X - 2b_3 Y + (2C_1 - 2C_3) Z + a_3^2 + b_3^2 + C_3^2 - C_1^2 = r_3^2 - r_1^2$

1 and 2 intersect forming a line. The direction of the line
may be found by taking the cross product of their respective
normals.

$(\emptyset)(2C_1 - 2C_3) - (2C_1 - 2C_2)(-2b_3) i ==> (2b_3)(2C_1 - 2C_2)i$
$+$

$(2C_1 - 2C_2)(-2a_3) - (-2a_2)(2C_1 - 2C_3) j ==> (-2a_3)(2C_1 - 2C_2) +$

$(2a_2)(2C_1 - 2C_3);$
$+$

$(-2a_2)(-2b_3) - (\emptyset)(-2a_3) K ==> (4a_2 b_3) K$

146

$$L = (2b_3)(2C_1 - 2C_2)$$

$$M = (2a_2)(2C_1 - 2C_3) - (2a_3)(2C_1 - 2C_2)$$

$$N = 4a_2 b_3$$

Once a point on this line is found, its equations may also be found.

1st sub $Z = \emptyset$ in 1.

$$(2C_1 - 2C_2)(\emptyset) - 2a_2 X = r_2^2 - r_1^2 - C_2^2 + C_1^2 - a_2^2$$

$$X = \frac{r_2^2 - r_1^2 - C_2^2 + C_1^2 - a_2^2}{-2a_2}$$

Now sub both x and z into 2.

$$-2b_3 Y = r_3^2 - r_1^2 - a_3^2 b_3^2 - C_3^2 + C_1^2 + 2a_3(X)$$

$$Y = \frac{r_3^2 - r_1^2 - a_3^2 - b_3^2 - C_3^2 + C_1^2 + 2a_3(X)}{-2b_3}$$

Now call the X, Y, Z XC, YC, ZC. Given this point the line equation is

3. $\quad \dfrac{X - XC}{L} = \dfrac{Y - YC}{M} = \dfrac{Z - ZC}{N}$

If A and 3. are intersected the two possible positions will be found

$$\frac{X - XC}{L} = \frac{Z - ZC}{N} \implies X - XC = \frac{L(Z-ZC)}{(N)} \implies X = \frac{L(Z - ZC)}{(N)} + XC$$

$$Y = \frac{M(Z - ZC)}{N} + YC$$

$$X^2 = \left(\frac{*LZ}{N} - \frac{LZC}{N} + XC\right)^2 \quad \text{where P will equal } \frac{-LZC}{N} + XC$$

$$Y^2 = \left(\frac{MZ}{N} - \frac{MZC}{N} + YC\right)^2 \quad \text{where Q will equal } \frac{-MZC}{N} + YC$$

plugging these into A we get:

$$\left(\frac{*LZ}{N} + P\right)^2 + \left(\frac{MZ}{N} + Q\right)^2 + (Z - C1)^2 = r1^2$$

$$\left(\frac{L^2}{N^2} + \frac{M^2}{N^2} + 1\right)Z^2 + \left(\frac{2LP}{N} + \frac{2MQ}{N} - 2C_1\right)Z + \left(P^2 + Q^2 + C_1^2 - r_1^2\right) = \emptyset$$

$$Zc = -B + \frac{B^2}{2}B - \frac{B}{}$$

we want to minimize

$$S = (Z_{ic} - Z_{io})^2 \quad \text{using } C_1, a_2, C_2, a_3, b_3, C_3$$

this is done by finding the six partials and moving the variables toward their min.

16. IMAGING SYSTEM

An additional phase of the EAVE program includes the development of an autonomous vehicle imaging system. By adding a real-time television link to the system a remote operator can monitor the functioning of the vehicle. This capability could be used to remotely control the vehicle or to visually inspect a submerged structure. The requirements of underwater imaging from an untethered vehicle are greatly influenced by the acoustic channel data capacity. Since the acoustic channel cannot support a high data rate, the source must be band-limited to make effective use of the narrow channel capabilities.

A typical television picture (home television) contains a large amount of information. If a one second TV signal were digitized to 8 bits the resulting data would require almost 60 million bits of storage. Since the acoustic channel has data capacities of 5-20 Kbits per second for ranges of 1000 feet, it would take over 50 minutes to transmit this block of data. Clearly modifications must be made to the video source before it can be applied to the ocean environment. Several immediate tradeoffs can be implemented to make a practical imaging system. In this research the frame rate was reduced from 20 to 2 frames per second. Also for many tasks a reduced resolution is feasible; typical television (in the U.S.) has 525 lines with 480 pixels per line. Research into ocean imaging is being conducted with a 100 x 100 pixel LCCD camera as the input device. A further data reduction can be made by using a reduced gray scale. Tests have shown that a 4 bit gray scale provides enough contrast and dynamic range for feature recognition. This is an additional possibility to reduce video data and make the system more useful in the ocean environment - bandwidth reduction by digital computer.

There are currently several computer algorithms which can be used to eliminate redundant information in digitized images. These algorithms vary in speed, complexity and compression ratio and must match the application. The algorithm chosen for this research is Micro Adaptive Picture Sequencing (MAPS), which is a two-dimensional spatial reduction technique. This algorithm has shown proven fidelity and compression ratios and promises compression ratios from up to 12:1 on low resolution ocean images.

The hardware configuration for this project is shown in Figure 69. This block diagram shows the relationship between various components of the imaging system. As mentioned earlier the input device is a CCD camera which contains a Reticon 100 x 100 element photo diode array with CCD transfer mechanism. This camera is able to sense an entire frame of data at one instance and read the individual pixels out serially. Another benefit of this CCD imaging system is the low light level capability of the sensor. This allows the camera to be used in the ocean and require less artificial light. The sensitivity of the camera is determined by the clock frequency which sets the light integra-
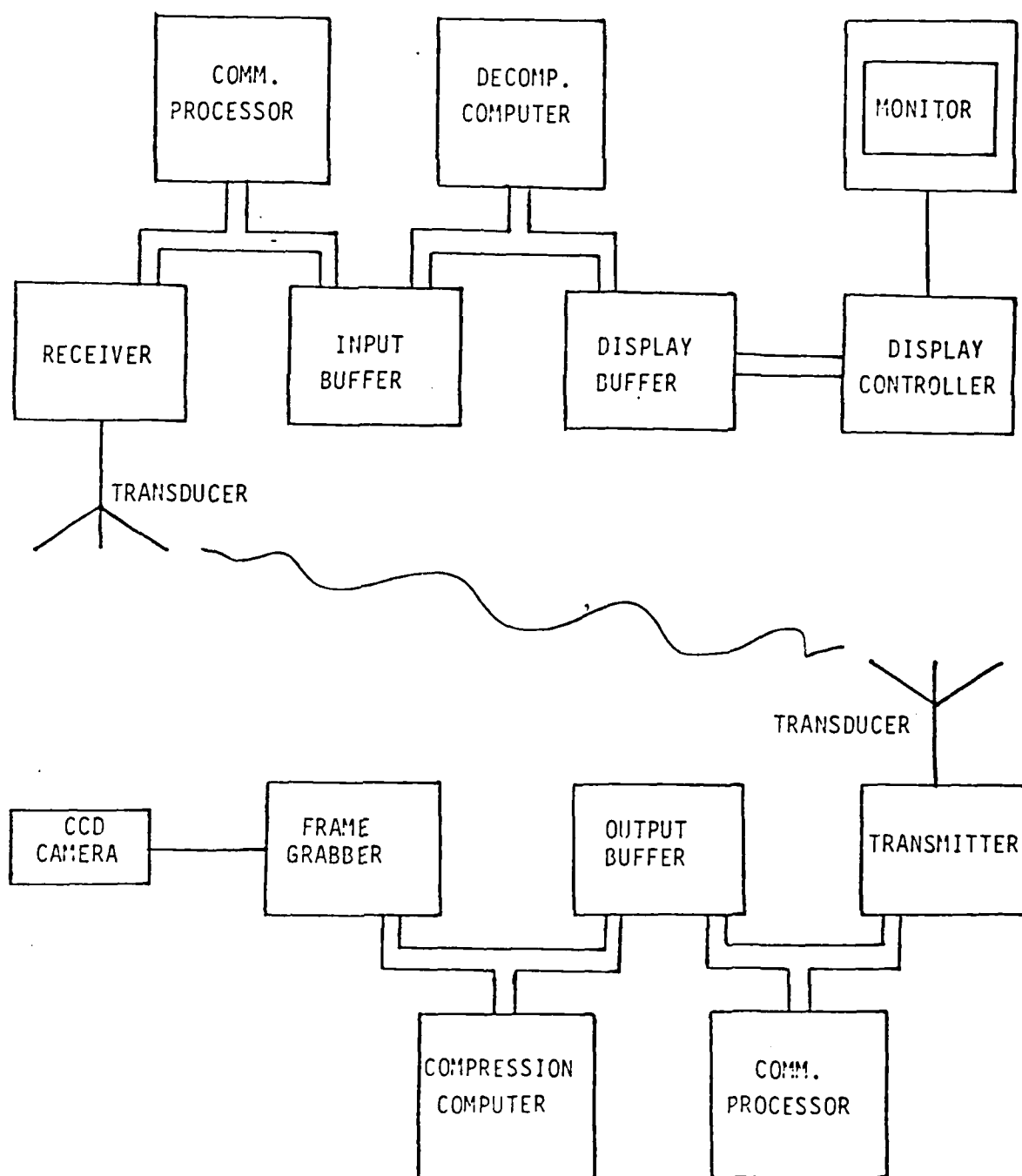
149

Fig. 69 Imaging System Configuration

tion time of the photo diode array. Currently a clock rate of 125 kHz produces over two frames per second of video with typical room lighting.

The data compression and decompression CPU's are the core of each subsystem. The Motorola 68000 microprocessor was chosen for this application because of its compatibility with other vehicle components and its computing power. Each CPU runs the dedicated task of image manipulation while special purpose hardware handles the input and output of the image. After the CCD camera senses the image the frame grabber digitizes and stores an entire frame of data without supervision from the compression computer. This architecture makes efficient use of the available resources by allowing the 68000 to concentrate on image compression. Once the image is compressed the output data is placed in an output buffer where it awaits error correction coding and transmission. The communications processors supervise the tasks of image transmission and receiving and error reduction. These devices are necessary to increase the system throughput and ensure image fidelity. At present the communications processors and the acoustic link have not been designed.

Once the image data is received the task of reconstructing begins. The communications processor passes data blocks to the 68000 which decodes and decompresses the image; image compression and decompression are inverse operations. After each data block is interpreted the information is passed to a special purpose output device; the Motorola 6845 a Cathode Ray Tube Controller (CRTC) which does the job of screen refresh. Since all video systems need a high refresh rate to prevent flicker, the job is dedicated to the CRTC. This device presents the display monitor with the appropriate analog video information along with the sync signals used in standard television.

The above hardware configuration gives the system flexibility and efficient operation in the processing of 100 x 100 images. But, the actual bandwidth compression algorithm is at the heart of the imaging process. As mentioned earlier MAPS is the algorithm chosen for use in this research. The algorithm seeks to reduce the total data content in an image by seeking to locate regions of redundant information. Redundant information refers to areas in the scene that have a constant or similar gray level on the 4 bit gray scale. A flowchart for the MAPS algorithm is shown in Figure 70. As can be seen the compression operates on independent 8 x 8 pixel blocks and continues throughout the image until all pixels have been processed. In the current application the image is sectioned into 12 rows of 12 blocks representing an image of 96 x 96 pixels. The additional 4 pixels on each row are discarded.

The MAPS technique starts by sampling a 2 x 2 pixel area, if these 4 pixels intensities are within a certain threshold they are averaged together and retained for further processing. If the gray values are uncorrelated they are output as single pixel entities. The goal is to produce larger aggregates of pixels
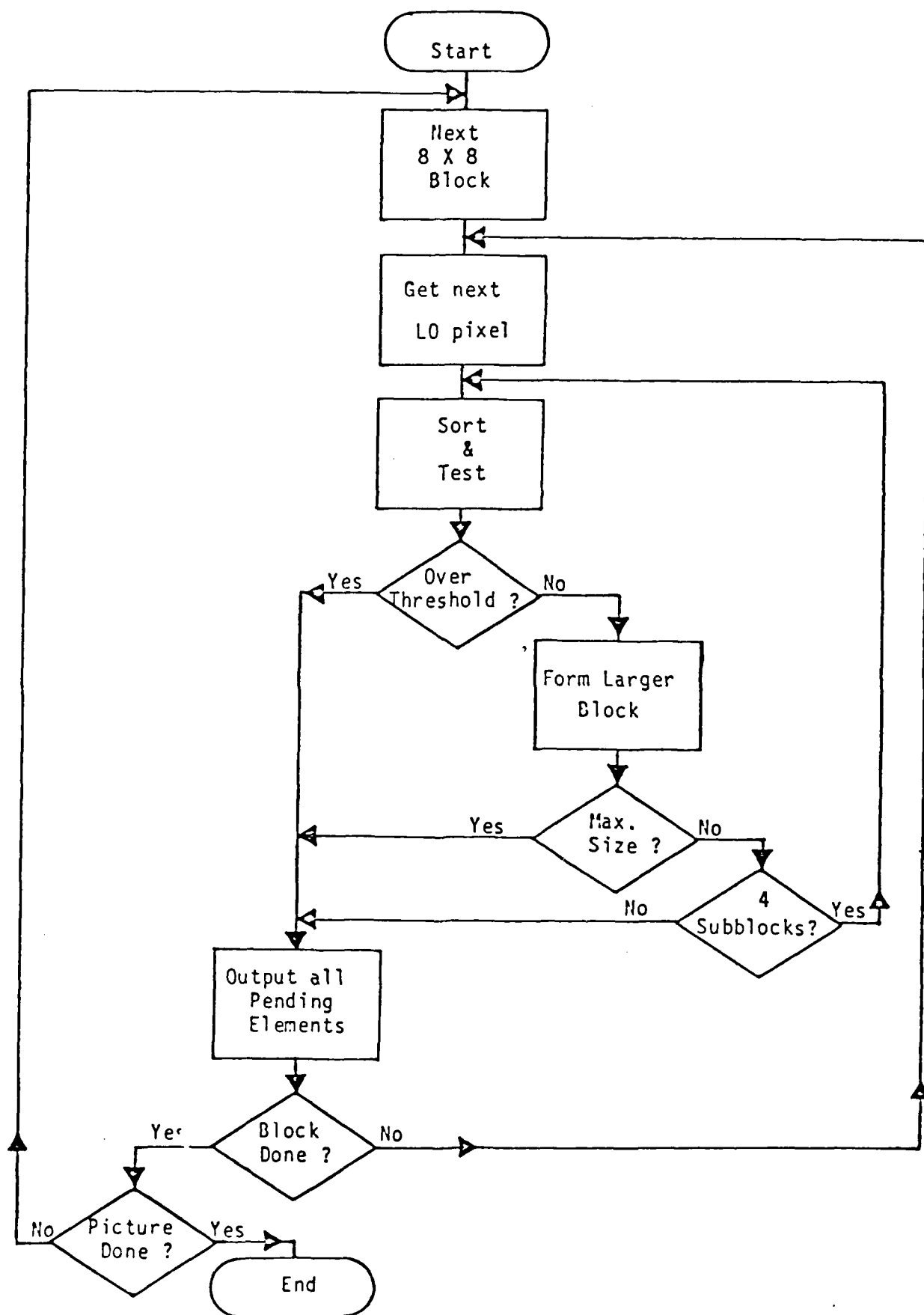
151

Fig.70 MAPS Flowchart

152

until the scene structure limits the combination because of a threshold violation. The input to the MAPS algorithm is a digitized image and the output is a data stream which represents the compressed image. This data stream contains two kinds of information; a resolution code and an intensity code (gray value). The resolution code represents the size of the combined pixel area; to limit overhead in this transmission scheme the output contains only 4 standard size blocks to which pixels may be combined. Only 2 bits of binary information are necessary to encode the 4 possible block sizes which always occur as square regions. The sizes are 1 x 1, 2 x 2, 4 x 4, and 8 x 8 pixels which are referred to as level 0 to 3 blocks (L0 - L3). Since most images exhibit high correlation between nearby pixels, data compression occurs as a result of the MAPS algorithm. Instead of sending 64 pixels of similar gray value a single resolution code which represents the 8 x 8 pixel block may be sent with the gray value of the whole block.

The input image to the compression process is a 100 x 100 pixel image which is digitized to 4 bits. This image has a data content of 40,000 bits before compression and the reduction technique is designed to operate on 96 x 96 pixel representing a data content of 36,864 bits per frame. A trial run of the MAPS compression process was run of the 68000 development station at the Marine Systems Engineering Lab. The input picture of the submarine Aluminaut was digitized from a standard vidicon camera and compressed on the development station. The results show data rates of 7.7K bits and 6.0K bits per image using different transmission schemes; the compression ratios are 4.8 and 6.1 to 1 for this detailed image. Simpler images will have higher compression ratios due to the lower information content.

Present simulations of the MAPS algorithm running in real time have shown the efficiency of the 68000 assembler programmed version. Timing trials have shown that a 4 MHz 68000 microprocessor can compress over 4 frames of data per second. This corresponds to an output data rate of 20 to 35K bits for typical images. The frame rate is dependent to an extent on the information content of the scene, but the acoustic channel is the major bottleneck limiting the transfer of data. For this reason it is necessary to match the frame output rate to the capacity of the channel.

Research into the error reduction potential of MAPS has revealed useful structures inherent in the algorithm format. Since a compressed frame must be reconstructed to an image of 96 x 96 pixel it is possible to detect errors in the resolution code which would be the most severe error. A loss of one resolution code could cause the entire scene structure to be destroyed. Any error correction scheme must carefully protect the resolution codes from corruption. In the same way that entire frame represents a fixed data content of 96 x 96 pixel it is proposed to divide the image transmission into sync intervals for the purpose of error correction and detection. A sync interval of two 8 x 8 pixel MAPS blocks represents a good choice between data

153

rate and error correction simplicity. After each area of 8 x 16 pixels is transmitted with its constituent resolution codes the sync mark is placed in the data stream. Upon decompression the processor checks that the reconstructed area represents an 8 x 16 pixel region. If not, the constraint equation is solved and the appropriate error is corrected.

$$L3 + 4 \times L2 + 16 \times L1 + 64 \times L0 = 8 \times 16 = 128$$

Above L0 to L3 refer to the number of the various resolution codes received during the previous sync interval. It is important to note that this correction scheme assumes errors in a single resolution code.

At this point of the research a few enhancements to the system as well as more development can be suggested. The most important point is the development and interface of the imaging system to a high data rate acoustic telemetry system. This would allow actual sea trials of the untethered imaging system. A related thrust in the communication process is the inclusion of dedicated communication processors to monitor the I/O and perform the error encoding and correction. It is likely that more elaborate error reduction schemes will be needed to enhance the system performance. Also there are many techniques to enhance image fidelity that should be studied in the context of ocean imaging. This effort would also lead into pattern recognition and smarter vehicles. Another enhancement in the area of image fidelity is the reduction of noise in the input hardware. A noisy image is unpleasing to view and severely limits the ability of a compression process to reduce data.

16.1 Results Summary

The MAPS compression algorithm has shown good results on typical ocean scenes. An imaging test bench has been constructed to digitize, compress, decompress and display the scene. Previously the operations were performed on a single CPU which simulated the compression and decompression steps. In order to more realistically demonstrate the imaging system two 68000 computer systems were joined by a 9600 baud serial link; this gave an effective data rate of 7.2 kilobits/second.

The compression ratio and hence the frame rate of the system are dependent on the matrix of contrast thresholds and the complexity of the image. The operator may manipulate the threshold matrix to achieve the desired frame rate but the scene detail remains a random variable. Trials have shown compression ratios of up to 15:1 on very simple scenes; the expected useful operating range of MAPS on low resolution images is from 4:1 to 8:1.

Photo A (Figure 71) is a snapshot of a joint on a submerged structure while Photo B shows the image digitized and displayed on a monitor. The MAPS compression routine was run on the

digitized scene and the decompression process was simulated. The
results shown in Photos C and D are in the following table.

|	Standard (bits)	Enhanced (bits)
Photo C	16,722	10,810
Photo D	9,630	6,854

The standard and enhanced trials refer to different trans-
mission schemes which can be used.   The standard format sends 2
bits of resolution code and a four bit intensity representation
while the enhanced format has Huffman codes for resolution and a
3 bit differential intensity representation.

Transmitting the image shown in photo D over the 9600 baud
line would take .95 seconds.   But, a frame rate of at least 2
frames/second is expected since the acoustic channel can support
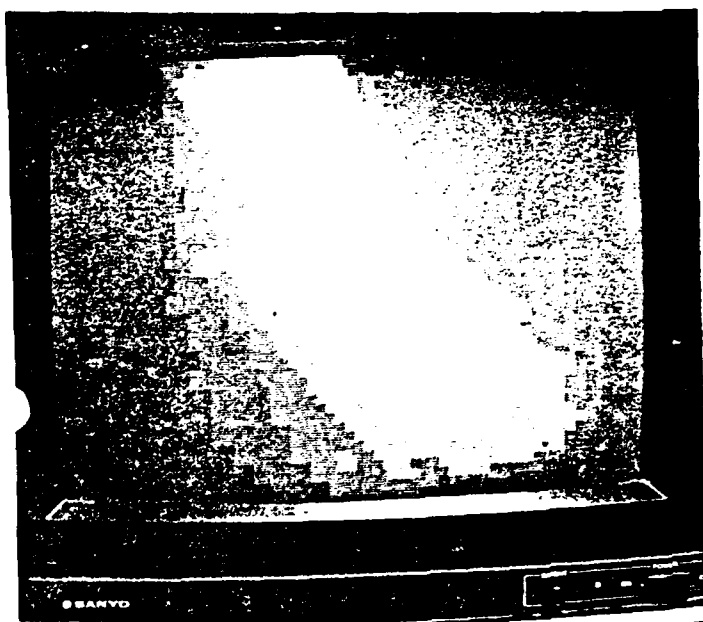more than 7.2 kbits/second over the desired ranges.
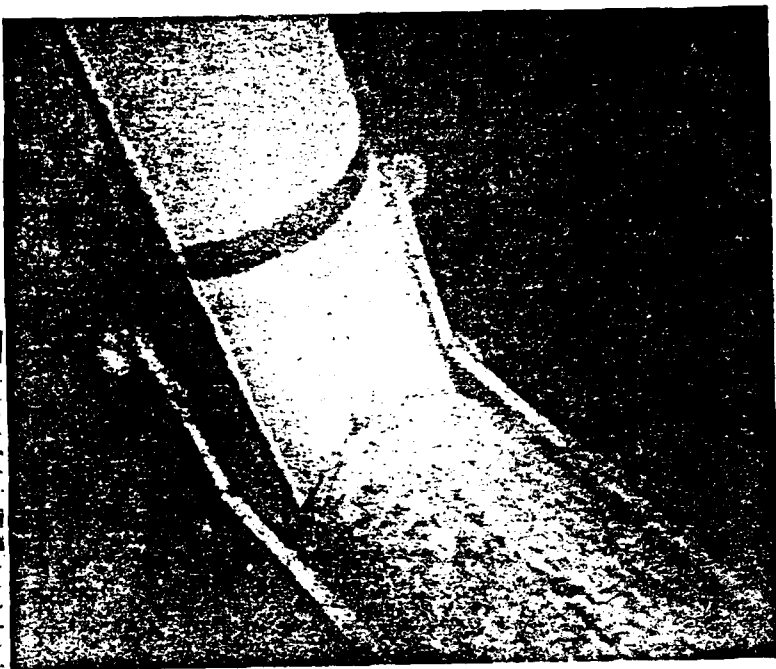
A   B
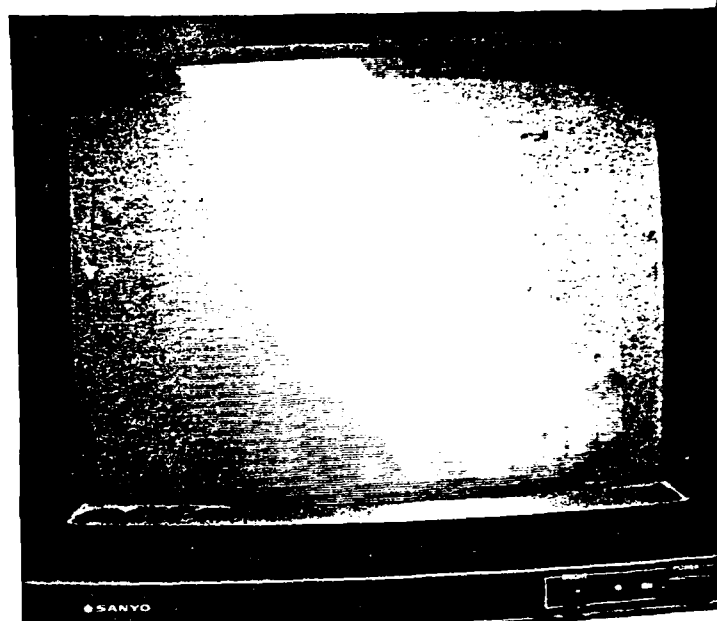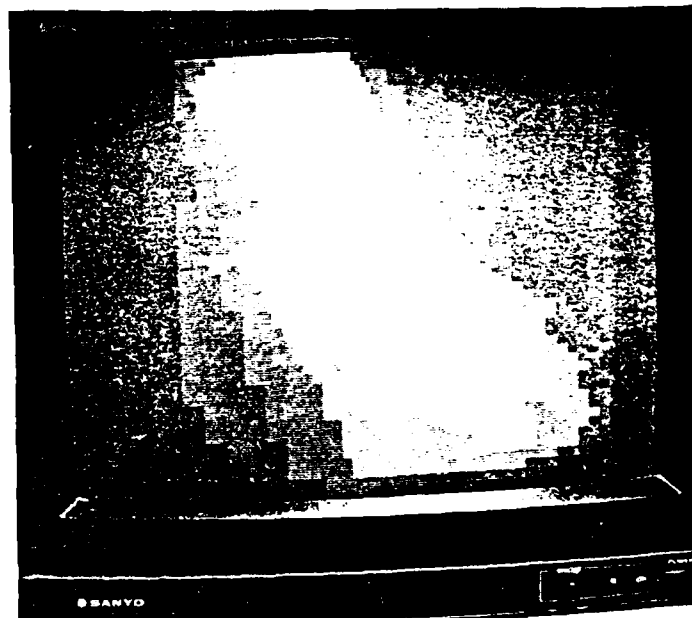

C   D

FIGURE 71:   IMAGING PHOTOS

17. ARCTIC INSPECTION MISSION SYSTEM (AIMS) SONAR

17.1 Description

The AIMS sonar sensor (Figure 72) is designed to be operated under ice. It is a six channel transceiver that will map the contour of the ice keel above the vehicle and monitor the depth of water beneath (Figure 73).

It consists of an array of five transducers directed upward at angles of 0, ±5 and ±10 degrees from vertical; the sixth is connected via a tether cable and can be oriented in any desired direction. All channels operate at 200 kHz, transmitting pulses of 350 microsecond duration at a rate determined by the user. Each beam pattern has 3db down points at ±3 degrees.

The entire assembly, except for the remote transducer, mounts in a cylinder eight inches in diameter and approximately seven inches high. This cylinder bolts to the top of the vehicle frame, aft of the compass. Electrical interface with the command computer applications card is via an 8 wire cable. Power and control signals are sent to the sonar which replies with a pulse indicating receipt of an echo. This delta time between transmit and receive signals corresponds to the distance between the vehicle and the surface generating the echo. Combining the distances from each transducer generates a map of the surface in a strip above the vehicle track.

The electronics are mounted on two boards within the cylinder described above. The transmitter board (Figure 74) generates the 200 kHz carrier frequency by dividing the output of a 2 MHz crystal. Pulse duration is determined by a monostable multivibrator, whose output is routed via a multiplexor to the selected output channel. The pulse gates a burst of carrier frequency to a pair of power FETs in a push-pull configuration, through a transformer to the transducer.

Since the transducers are reciprocal, the receiver is muted during the transmit pulse. The receiver has a 40db preamplifier for each channel. The control word from the command computer that selects the transmit channel also selects the output of the corresponding preamplifier for further filtering/amplification. A detector circuit triggers a comparator when the return signal threshold is crossed. The comparator output generates a pulse that is returned to the command computer signaling the completion of a transmit-receive cycle.

The command computer software determines the channel to be activated, and the pulse repetition rate.

17.2 Test Results

Initially, an operational test of the system as a whole was conducted by placing the remote transducer in a water tank,
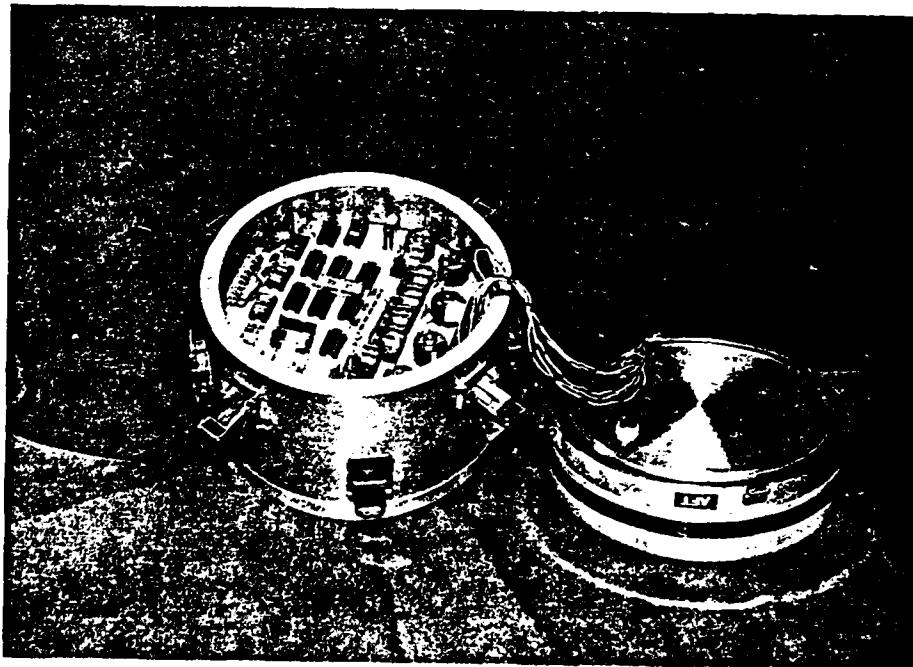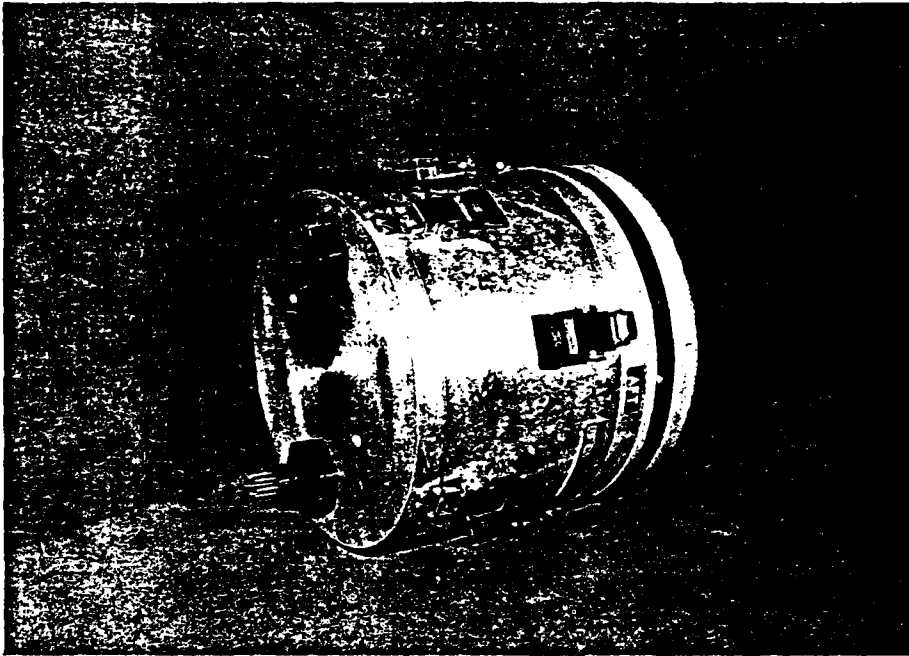
FIGURE 72: AIMS SONAR

SPECIFICATIONS

Operating Frequency:	200 kHz $\pm$50 Hz
Transmit Power:	300 Watts
Transducer Beam Angle:	$\pm3^{o}$ for ±3 db
Transducers:	One 5 element array with elements at 0°, $\pm10^{o}$ and $\pm20^{o}$ from vertical. One remote element connected with 6 feet of cable.
Receiver Sensitivity:	10 uv
Preamp Gain/Channel:	40 db
Receiver Bandwidth:	Approximately 10 kHz

Interface Connections

Channel Select Input:	Selects desired channel to transmit/receive on three binary lines, +6 volt true.
Start Input:	0 to +6 volt key pulse triggers transmitter
Stop Output:	0 to +6 volt pulse at time of received echo

FIGURE 73

FIGURE 74: TRANSMITTER BOARD

transmitting in that channel and verifying that returns were received and a STOP signal generated.

Next, the receiver and transmitter circuit boards were removed from their cylinder housing and mounted on a test stand for bench testing.

17.2.1 Transmitter

1. The following circuits were verified as operating correctly:
   a. 6V and 12V power supplies
   b. Channel select logic
   c. Start logic.

2. The quiescent current draw was measured to be approximately 50 miliamps. Response of current draw vs. pulse repetition rate was measured and is shown in Figure 75.

3. The transmitted wave form from each channel was observed. A photograph of the waveform from a typical channel is shown in Figure 76.

17.2.2 Receiver

1. Pre-amplifier gain was measured to be a nominal 40db. Table 7 presents the measurements from each channel.

2. Delays from application of a suitable pulse on the input to generation of a step pulse was measured. Statistics are shown in Table 8.

3. There are 5 potentiometers in the receiver circuit, excluding those in the power supply circuit. The role of each is outlined below.
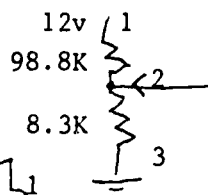
R19 (100 Ω) + in balance on U8
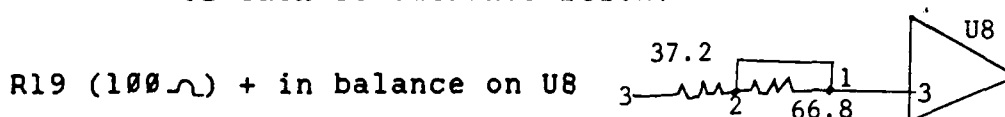
R28 (100K) voltage divider on 13600 (U9) Pin 1
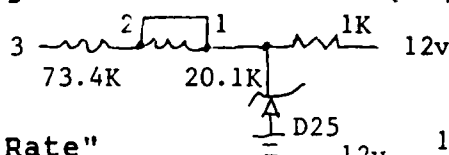
R29 (100K) "TVG Rate"
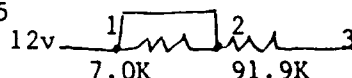
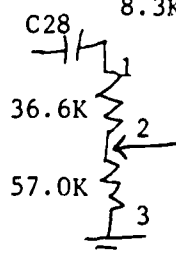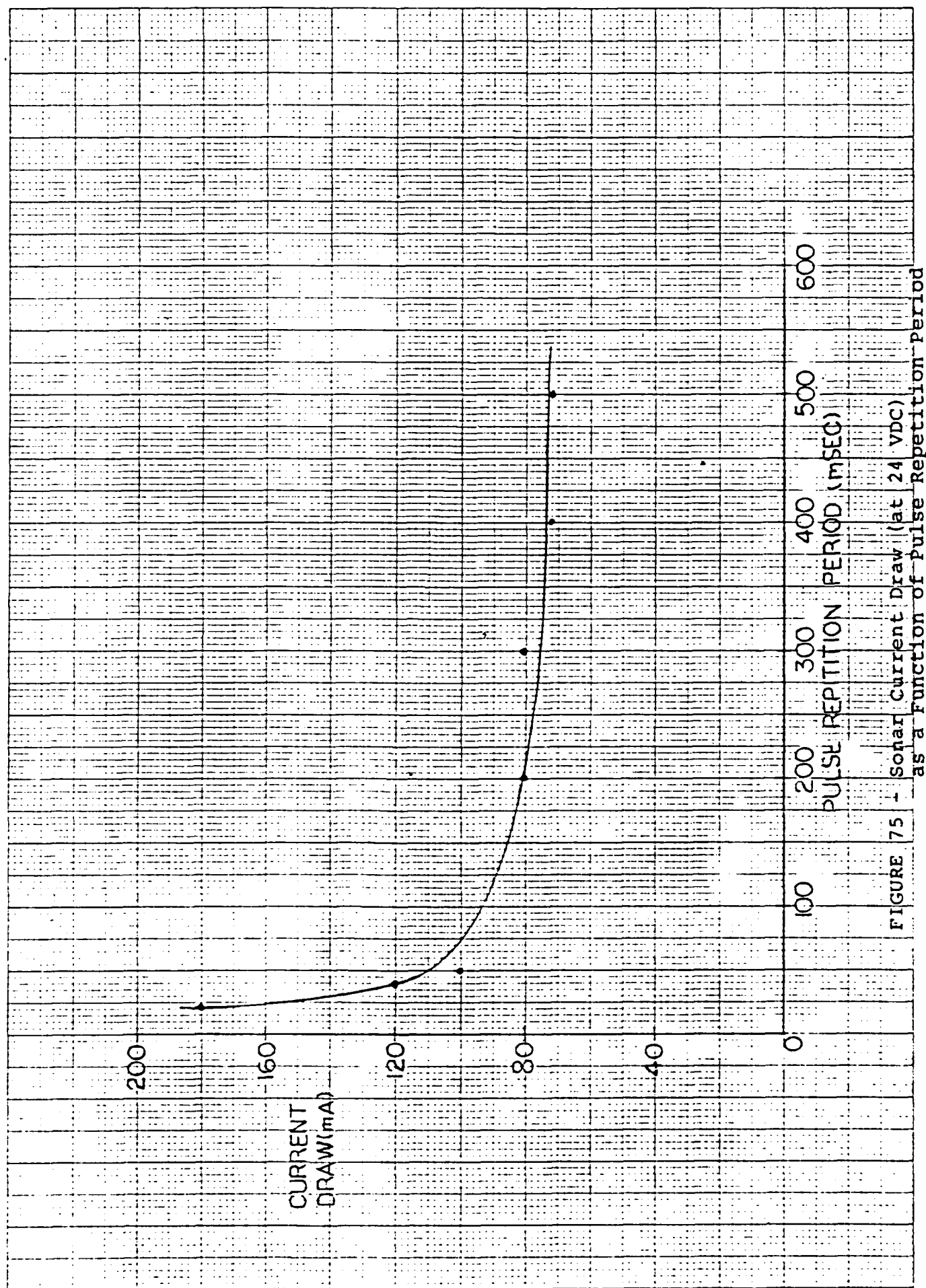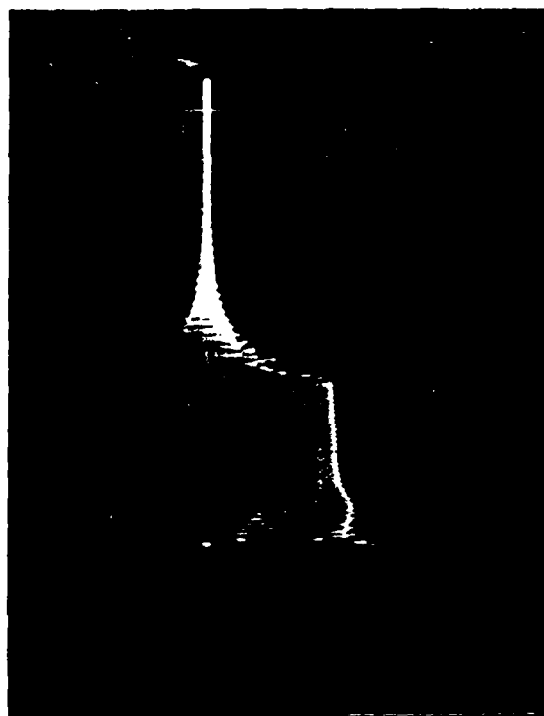R49 (100K) U9 balance

R50 (100K) U9 gain

161

FIGURE 75 — Sonar Current Draw (at 24 VDC) as a Function of Pulse Repetition Period

162

Output transformer secondary connected to transducers via $\approx$ 3 ft. cable

Negative Waveform

Positive Waveform

Vertical = 200 volts/cm
Horizontal = 100 μsec/cm
Pulse rep. rate = 2/sec

FIGURE 76: TRANSMIT PULSE, POSITIVE & NEGATIVE WAVEFORMS

163

TABLE 7

RECEIVER PRE AMPLIFIER GAINS

Channel	Pre Amp (1)	INPUT mV/pk to pk	OUTPUT volts, pk to pk	dB (2)
0	U5	3.16	0.38	41.6
1	U1		0.40	42.0
2	U2		0.40	42.0
3	U3		0.40	42.0
4	U4			42.0
5	U5			42.0

(1) U designator corresponds to identification in wiring diagrams.

(2) Typical calculation in Channel 0

$$dB = 20 \, Log_{10} \, \frac{Vout}{Vin}$$

$$dB = 20 \, Log_{10} \, \frac{.38 \, volts}{3.16 \times 10^{-3} volts}$$

$$dB = 20 \, Log_{10} \, (.12 \times 10^3)$$

$$= 20 \, Log_{10} \, (120)$$

$$= 20 \, (2.08)$$

$$dB = 41.6$$

TABLE 8

RECEIVER DELAY TIMES

Channel	0	1	2	3	4	5
Orientation	0°	$P10^o$	$S10^o$	$P20^o$	$S20^o$	remote
Mean	.132	.132	.117	.121	.116	.116
STD Dev	.012	.010	.008	.007	.007	.008
RMS	.133	.132	.117	.121	.116	.116

Notes:

(1) All data are in milliseconds
(2) Statistics calculated from 20 samples per channel
(3) Orientation, $P10^o$ = transducer whose center line tilted
     10° from vertical, on the port side of the array.
     S = starboard side
(4) Mean = $\overline{x}$ = $\dfrac{1}{N} \displaystyle\sum_{i=1}^{N} x_i$

$$\text{Std Dev = standard deviation} = \left[\frac{1}{N} \sum_{i=1}^{N} (x_i - \overline{x})^2 \right]^{\frac{1}{2}}$$

$$\text{RMS = root mean square} = (\text{Mean}^2 + \text{std dev}^2)^{\frac{1}{2}}$$

4. Adjustment of the U9 gain potentiometer, R50, has a significant impact on receiver sensitivity. A graph of that affect is shown in Figure 77.

17.2.3 Tank Testing

A number of tests were run with the array of transducers submerged in a test tank of concrete construction whose approximate dimensions are 8 ft. x 6 ft. with 5 ft. of water in it. Multipath and backscatter and the tank's relatively small dimensions frustrated any attempt to obtaining meaningful performance data in terms of accurately measuring distances. However, one salient feature of the system did emerge and that is the saturation of the receiver preamplifier stage when a pulse is transmitted. The preamplifier output was observed to oscillate from rail to rail (i.e. the positive supply voltage and ground) for some 9 miliseconds before reaching a steady state. This precludes reliable performance at distances less than 20 feet. Various combinations of potentiometer settings in the receiver allow one to select the point at which the decision threshold is crossed. Thus, while possible to generate a STOP pulse at shorter distances, any such measurement is ambiguous.

17.2.4 Lake Winnipesaukee Testing

Testing from a barge on Lake Winnipesaukee consisted of three configurations.

1. Measuring depth from array to the bottom. Typical data are presented in Table 9.

2. Measuring depth from the array to the surface. Significantly more scatter was noted in these data. The returns appeared to be clustered about two distances - one at the surface, and one about 1.5 feet below. This phenomenon may be attributable to thermal layering near the surface. Those returns that exceeded three standard deviations from the mean value were deleted. The statistics from this modified data set are presented in Table 10.

3. Measuring horizontal distances. This last category was least successful. In almost all cases, returns were received earlier than expected. Analysis of the geometry involved leads one to conclude that echos are being received from the surface and bottom from energy in the beam side lobes. These side lobes are in the range of 10 degrees to 20 degrees off the transducer centerline. In the case of 0 degree transducer, the return received may be the interval between the n + 1st transmit signal and the nth return. That is, from two different pulses. The pulse repetition rate was 1 pulse per second while a travel time of 1.2 seconds was possible.
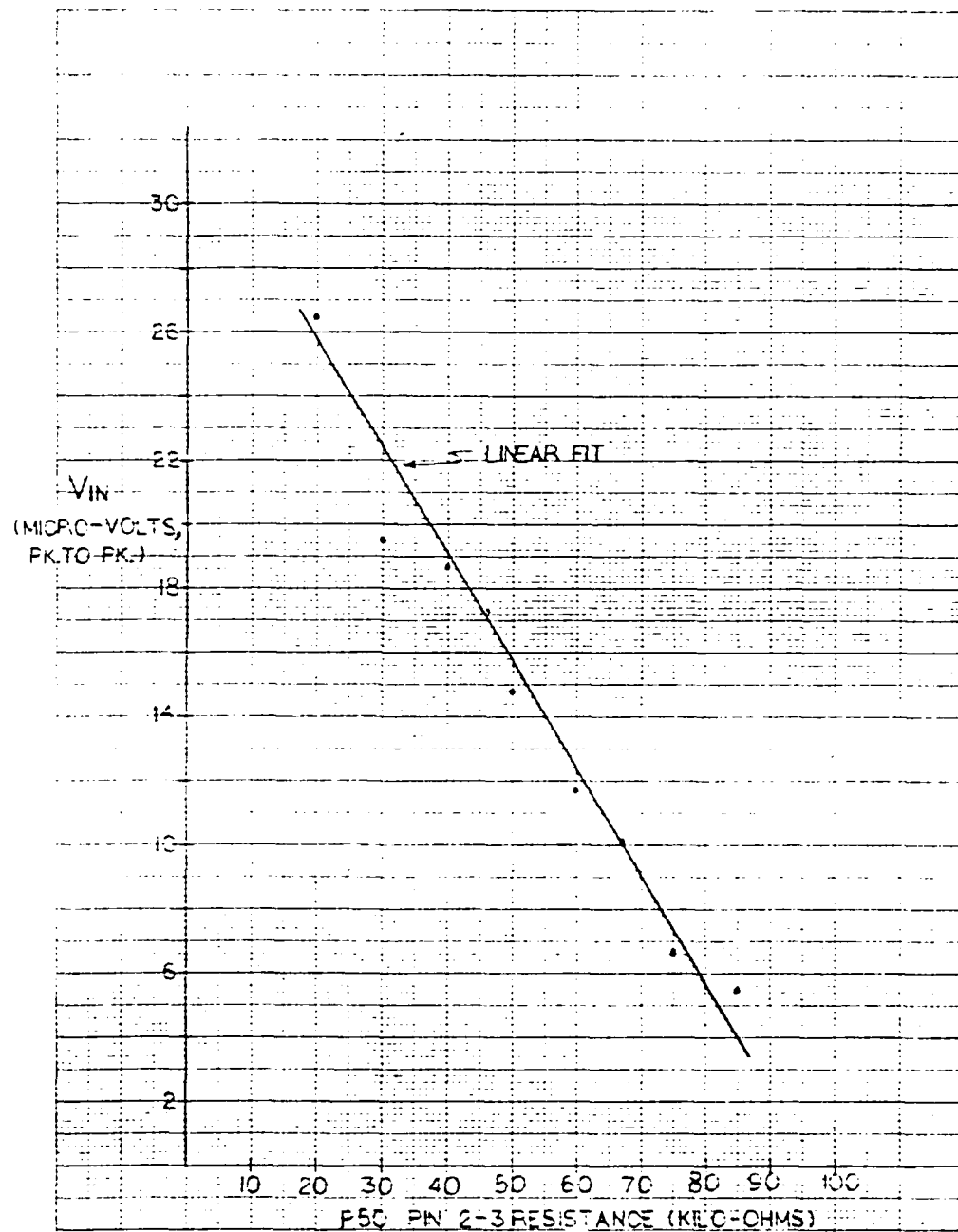
FIGURE 77 - Vin Calculated Assuming a Simple Voltage
Divider Action, With the Transmitter having
Infinite Impedance $Vin = 7.8 \times \left(\dfrac{VSG}{1 \text{ meg}}\right)$

167

TABLE 9

DEPTH SOUNDINGS

Channel	0	1	2	3	4
Orientation	0°	P10°	S10°	P20°	S20°
Travel Time (milliseconds)					
Mean	15.526	15.482	15.371	15.686	16.311
Std. Dev.	0.192	0.128	0.159	0.083	0.260
RMS	15.528	15.483	15.372	15.686	16.313
One Way Distance (ft)	37.14	37.03	36.77	37.52	39.02
Reference (ft)	36.75	37.32	37.32	39.11	39.11
Error (ft)	+0.39	-0.29	-0.55	-1.59	-0.9
Error (%)	1	-0.8	-1.5	-4.0	-0.2

NOTES:

(1) Sound velocity 4.784 feet per millisecond, in fresh water,
     58°F temperature, 0 depth calculated as average of four
     methods presented in Urick's "Underwater Sound for Engineers".

(2) Travel times corrected with channel delays from Table 2.

TABLE 10

DISTANCE MEASUREMENT TO SURFACE

Channel	0	1	2	3	4
Orientation	0°	$P10^{\circ}$	$S10^{\circ}$	$P20^{\circ}$	$S20^{\circ}$
Travel Time (milliseconds)					
Mean	9.868	10.138	10.185	10.041	10.453
Std. Dev.	0.147	0.048	0.150	0.107	0.041
RMS	9.869	10.138	10.186	10.042	10.453
One Way Distance (ft)	23.61	24.25	24.36	24.02	25.00
Reference (ft)	24.75	24.37	24.37	25.54	25.54
Error (ft)	-1.14	-0.12	+0.01	-1.52	-0.54
Error (%)	-5	-0.5	0	-6	-2

NOTES:

(1) Sound velocity 4.784 feet per millisecond.

(2) The returns off vertical are assumed to be reflections
     from the barge's flotation tanks.

(3) Travel times adjusted for channel delay.

17.3 Conclusions

The system is certainly operable as it is currently configured. One should expect distance measurements to be accurate to 1% to 3% at ranges up to 50 feet. The accuracy may very well improve at larger ranges. The system should not be employed at ranges less than 25 feet. Pulse repetition rate should not exceed 1 pulse per second. Note that the desired rate is a function of distance to be measured since more than one pulse should not be in the water at any one time.

17.4 Recommendations

17.4.1 Design

1. **Electronic** - consider muting the preamplifiers when transmitting a preclude saturation. The time varying gain stage should be reviewed. If the existing technique is indeed optimum, has it been implemented properly?

2. **Mechanical** - When the electronics boards are made as printed circuits, they should have a quick disconnect feature at all interface points. The receiver board should be accessible when the cylinder is opened, rather than the transmitter board, as it is now. Finally, the current method of bolting the boards to the cylinder end cap may not be satisfactory, particularly in view of mission length.

3. Software to process return times to generate contour surfaces.

17.4.2 Testing Recommended

1. Test at greater ranges with accurate reference.

2. Can a salient profile be identified? Determine relative accuracy as well as absolute accuracy.

3. Measure radiated acoustic pressure.

4. Determine transducer receiving sensitivity (i.e. volts generated per unit of pressure impacting its surface).

5. Run under computer control.

6. Run under the ice.

APPENDIX A

,

COMMAND SPEED -VS- SHAFT SPEED

SHAFT SPEED (RPM)

COMMAND SPEED

• 1.2°C
× 24°C

A-2
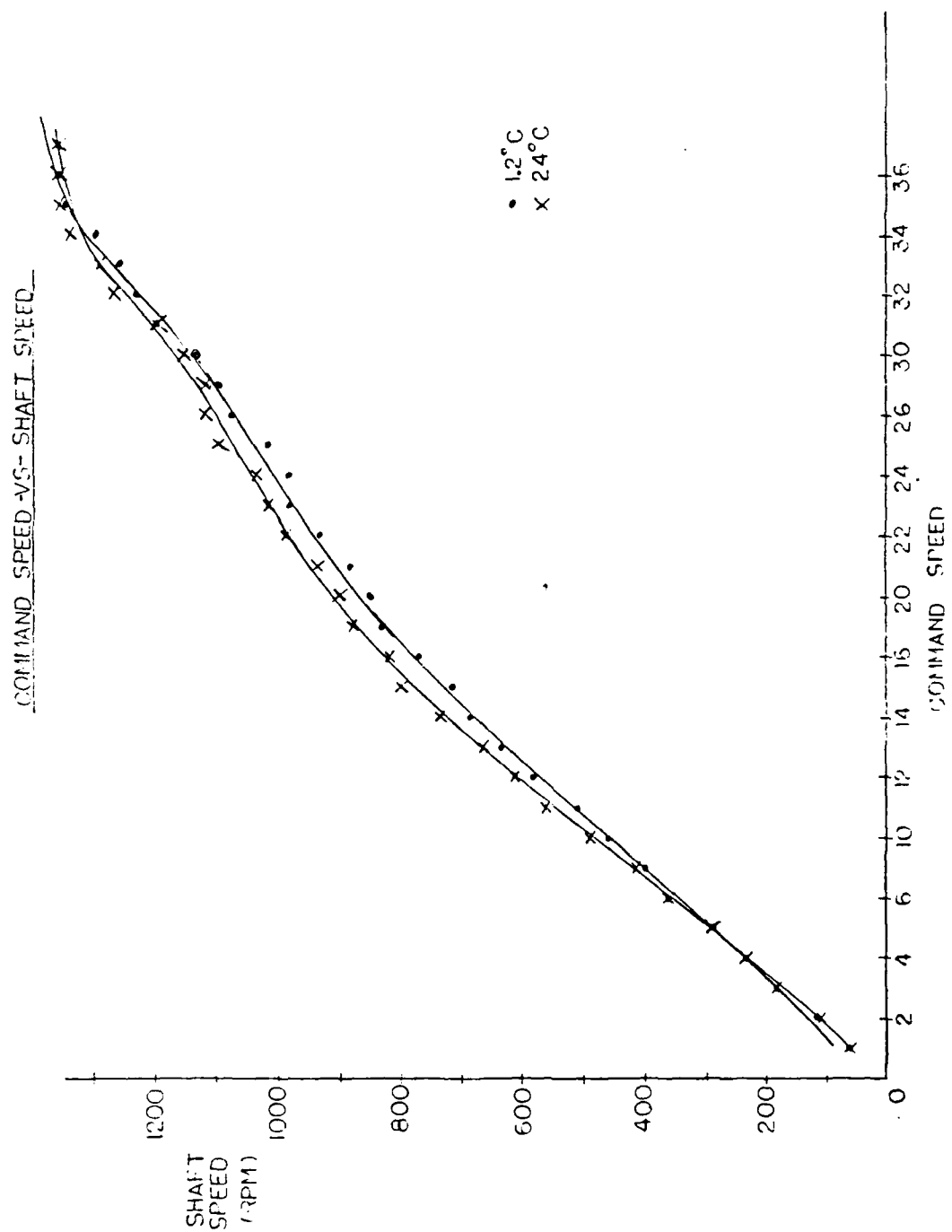
THRUST-VS-COMMAND SPEED [LEFT-HAND BYRD PROP]
UNMARKED THRUSTER

• FORWARD
× REVERSE

THRUST(LB)

COMMAND SPEED

A-3
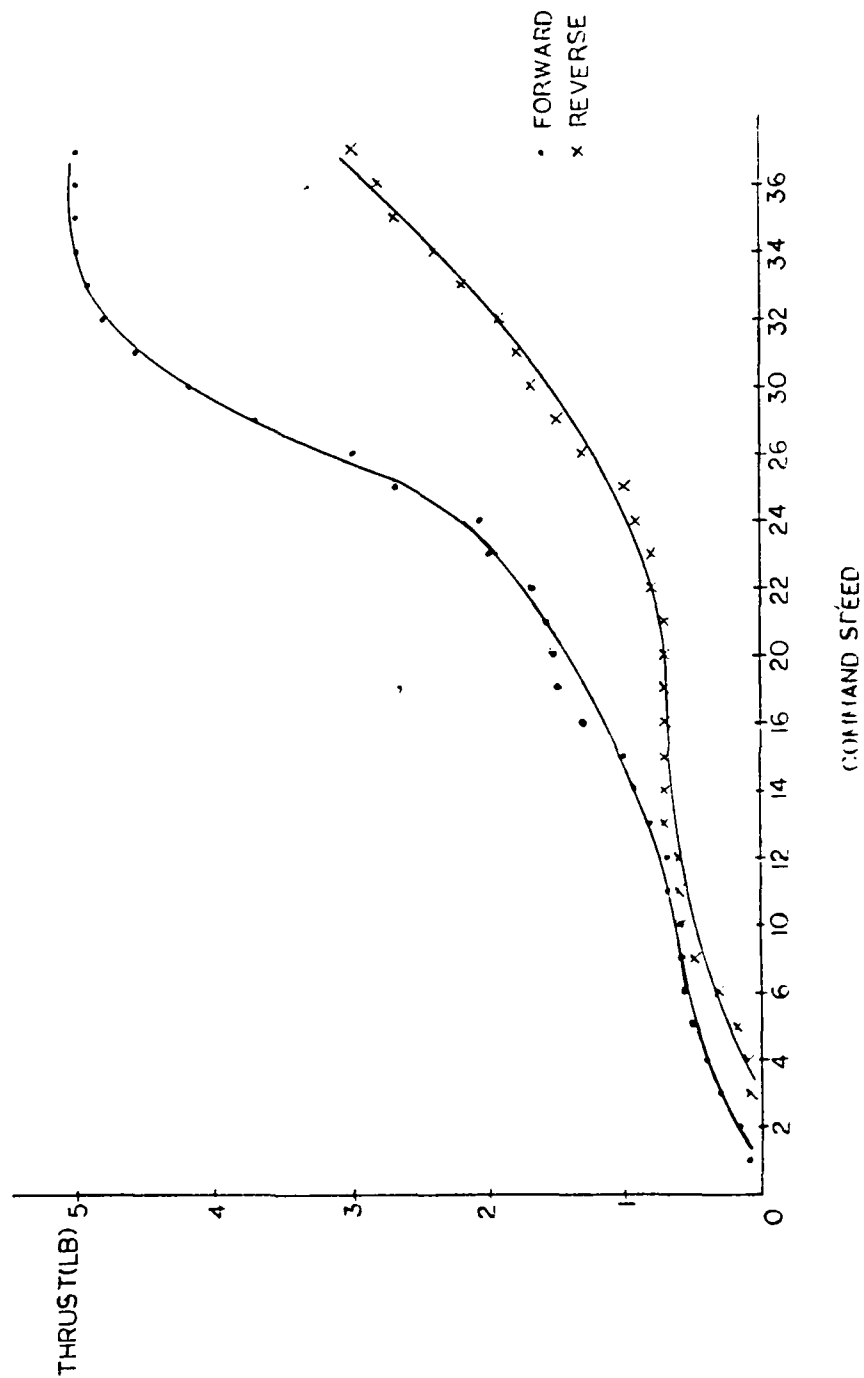
THRUST-VS-COMMAND SPEED [RIGHT-HAND MINN-KOTA PROP]
UNMARKED THRUSTER-AFTER CLEANING
FORWARD DIRECTION

THRUST(LB)
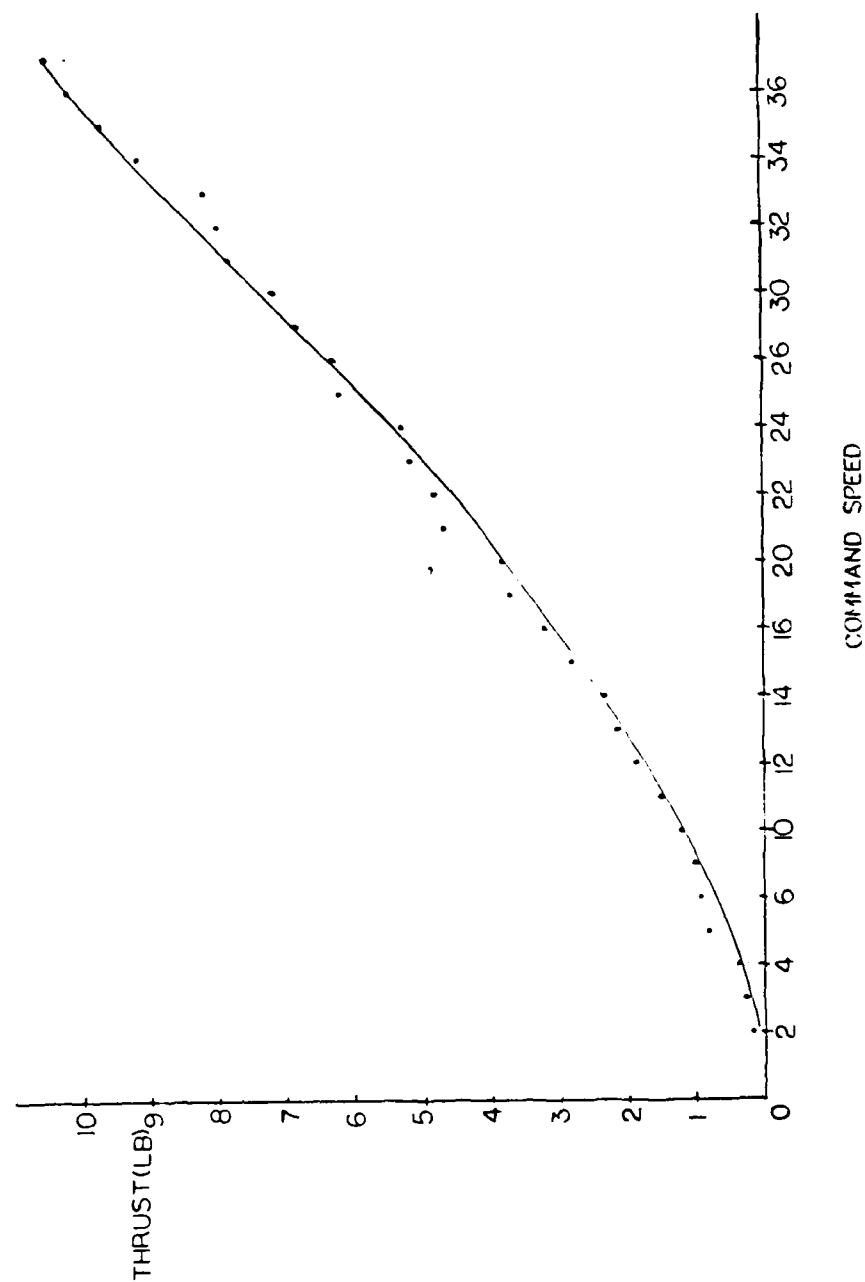
COMMAND SPEED

THRUST-VS-COMMAND SPEED
THRUSTER #1
REVERSE DIRECTION

THRUST(LB)

COMMAND SPEED

• KORT NOZZLE
× MINN-KOTA PROF **
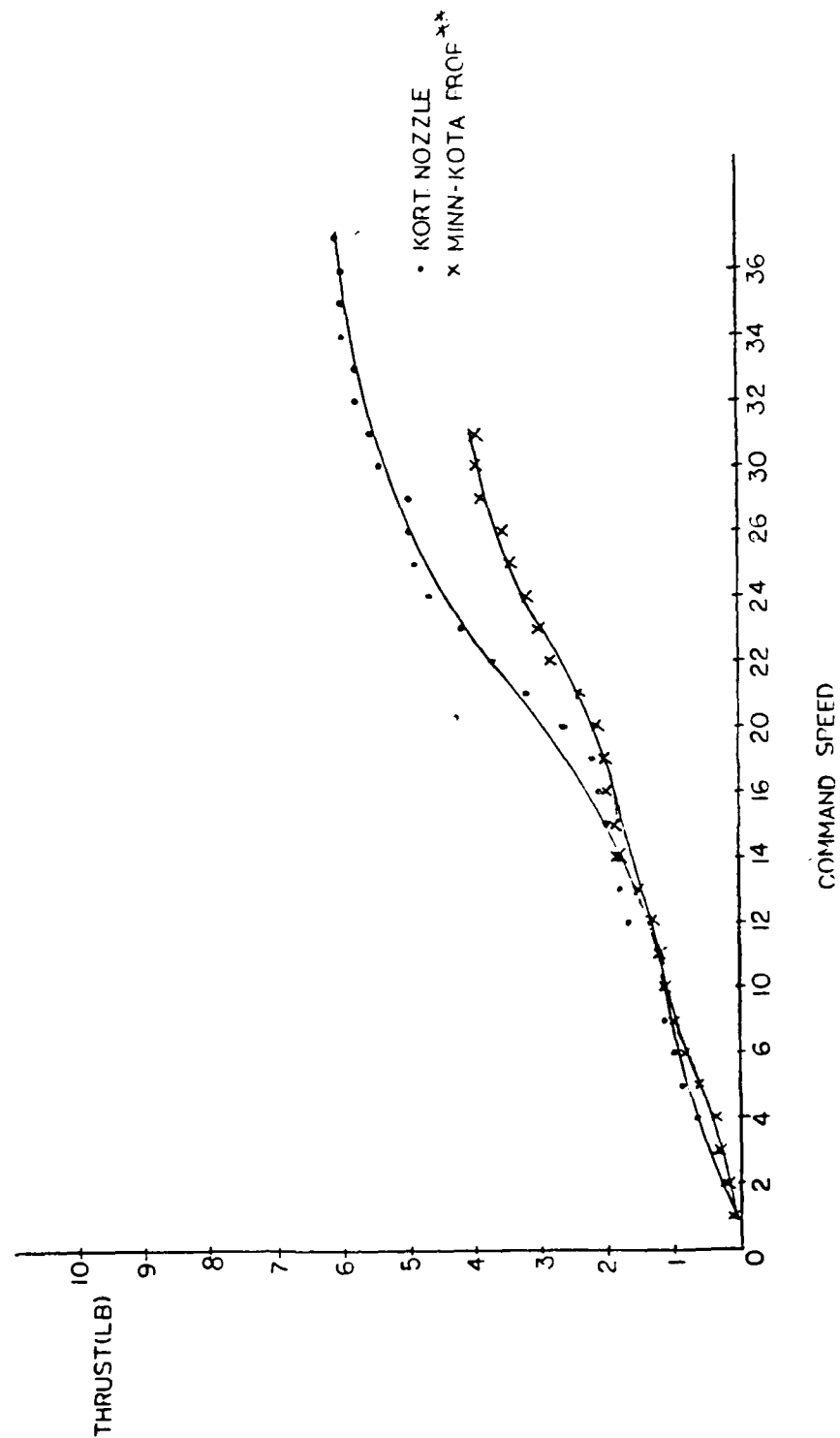
** SEE DATA

THRUST-VS-COMMAND SPEED
THRUSTER 1
FORWARD DIRECTION

• KORT NOZZLE
× MINN-KOTA PROP

THRUST(LB)
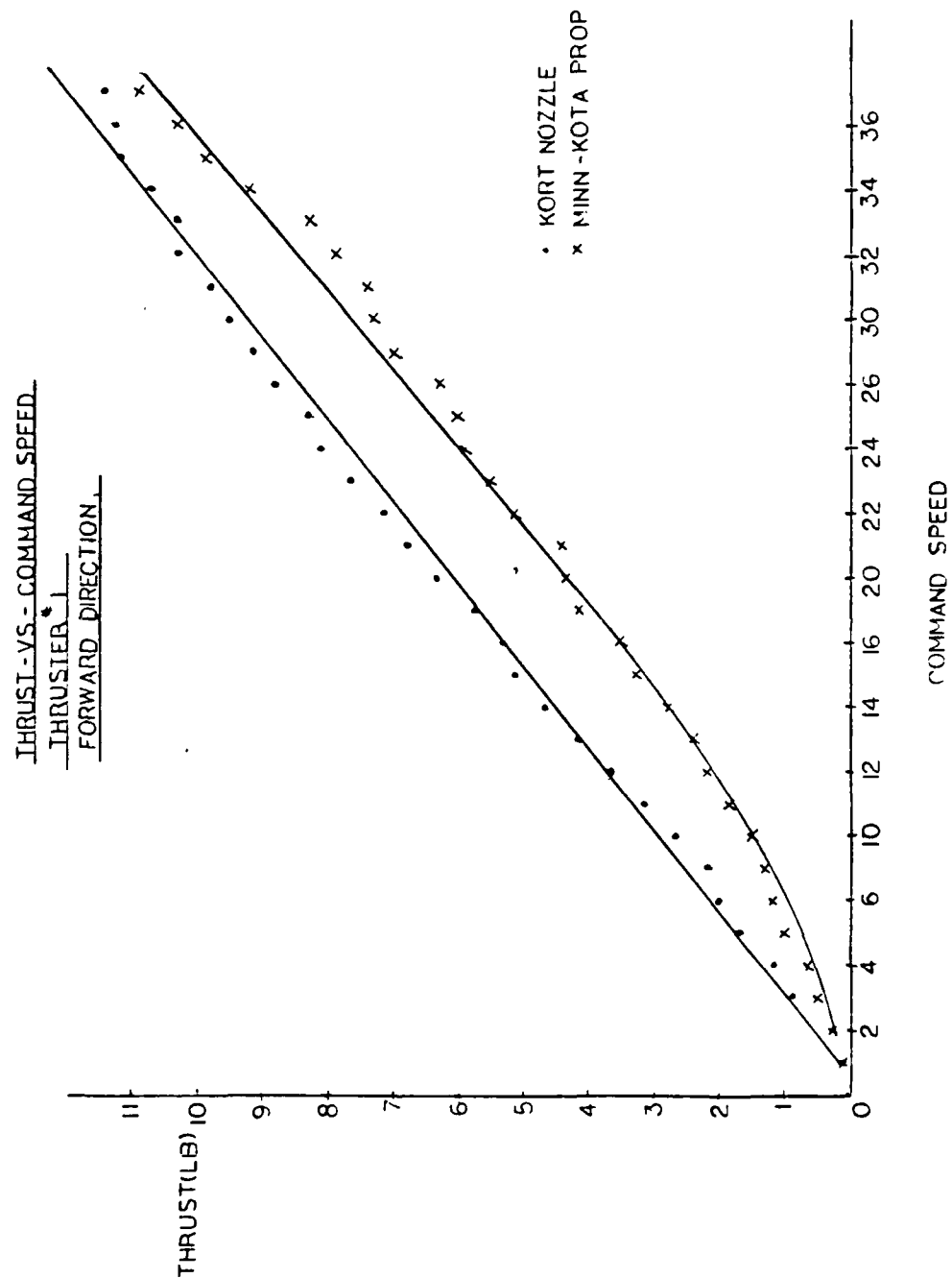
COMMAND SPEED

THRUST-VS-COMMAND SPEED [KORT NOZZLE]
THRUSTER #6

• FORWARD
x REVERSE

THRUST(LB)
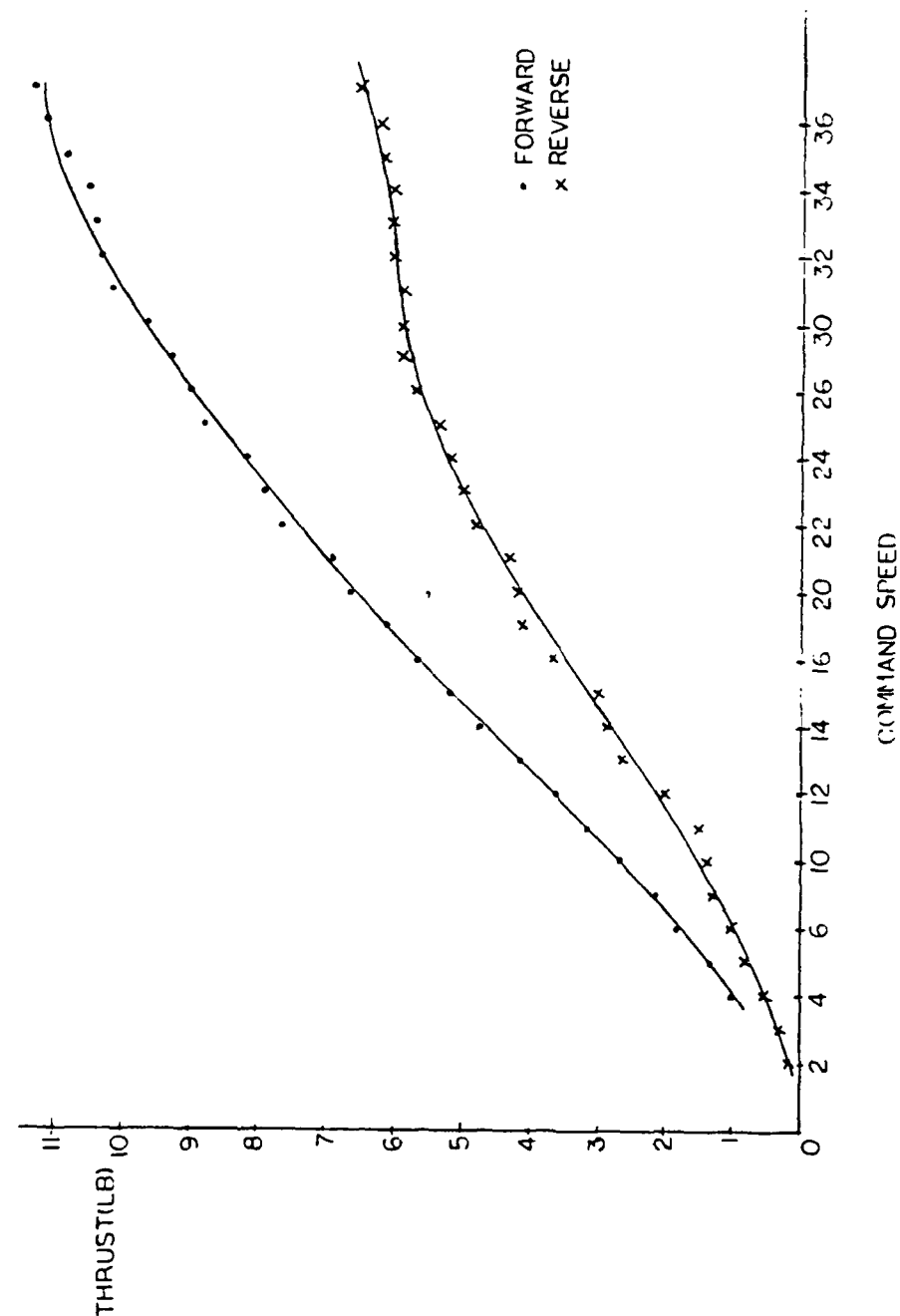
COMMAND SPEED

THRUST-VS-COMMAND SPEED
THRUSTER #5

THRUST(LB)

COMMAND SPEED

• FORWARD
× REVERSE

A-8

THRUST-VS-COMMAND SPEED
THRUSTER #4

FORWARD
× REVERSE
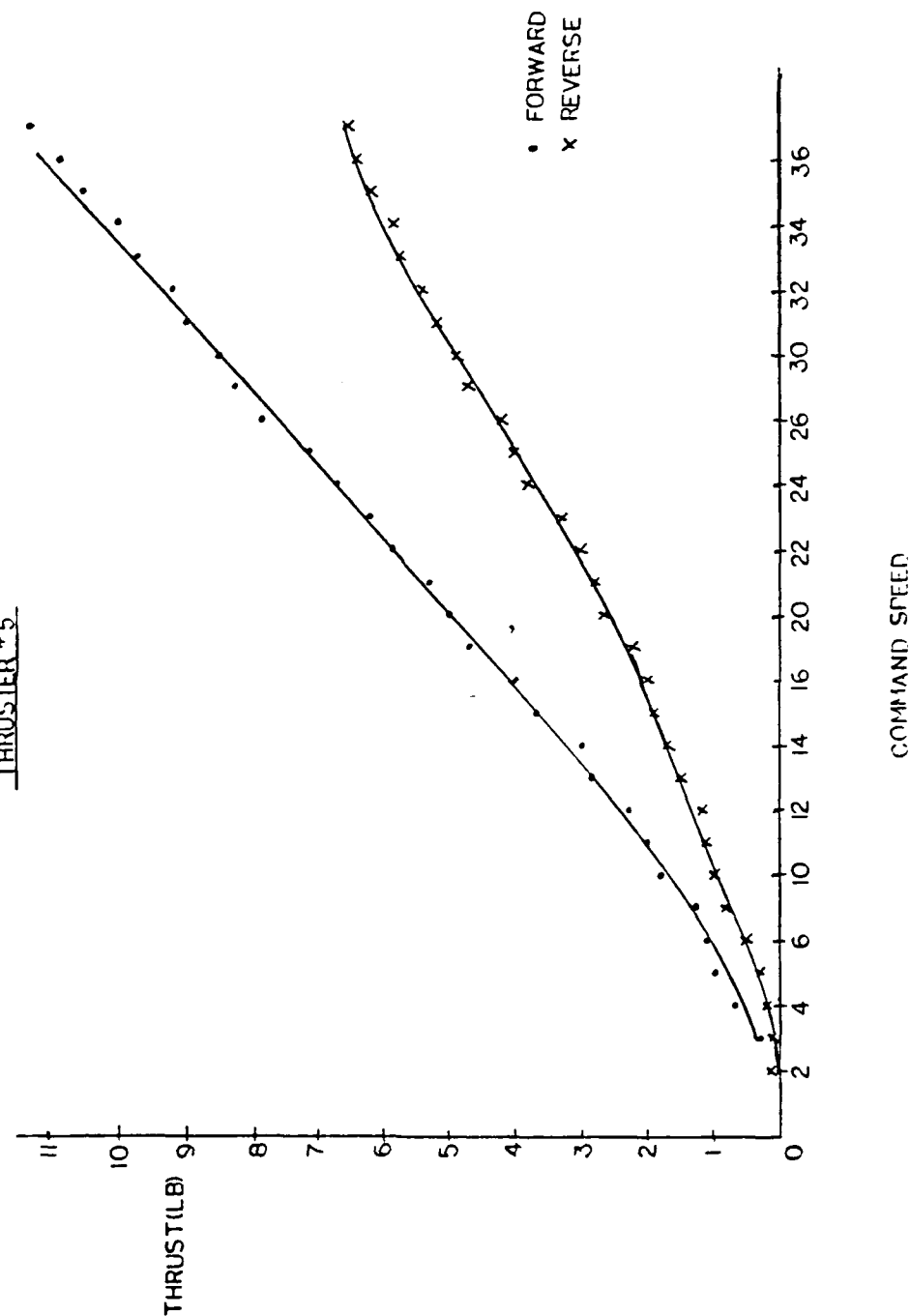
THRUST (LB)
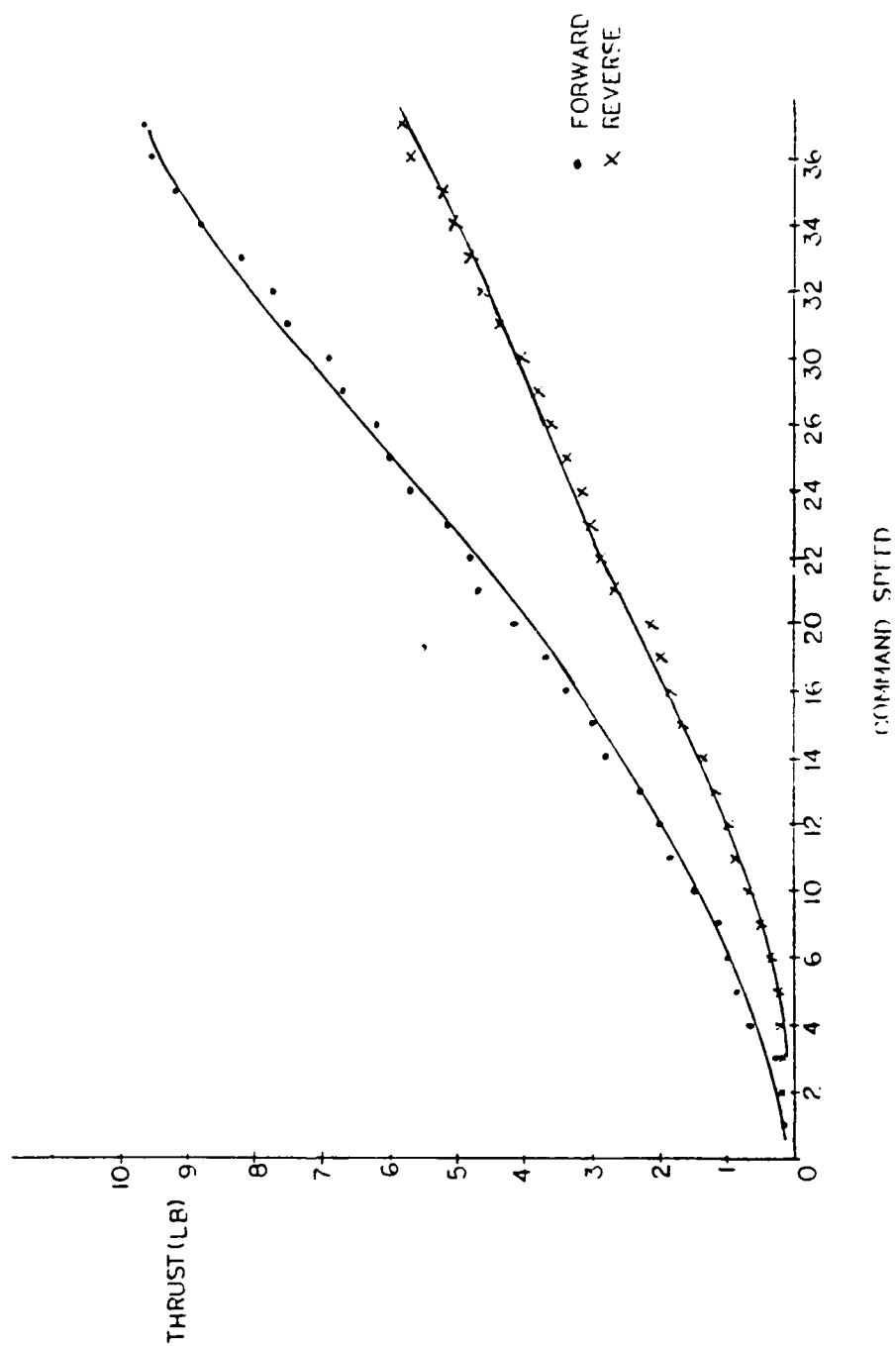
COMMAND SPEED

THRUST-VS-COMMAND SPEED
THRUSTER #3

THRUST (LB)
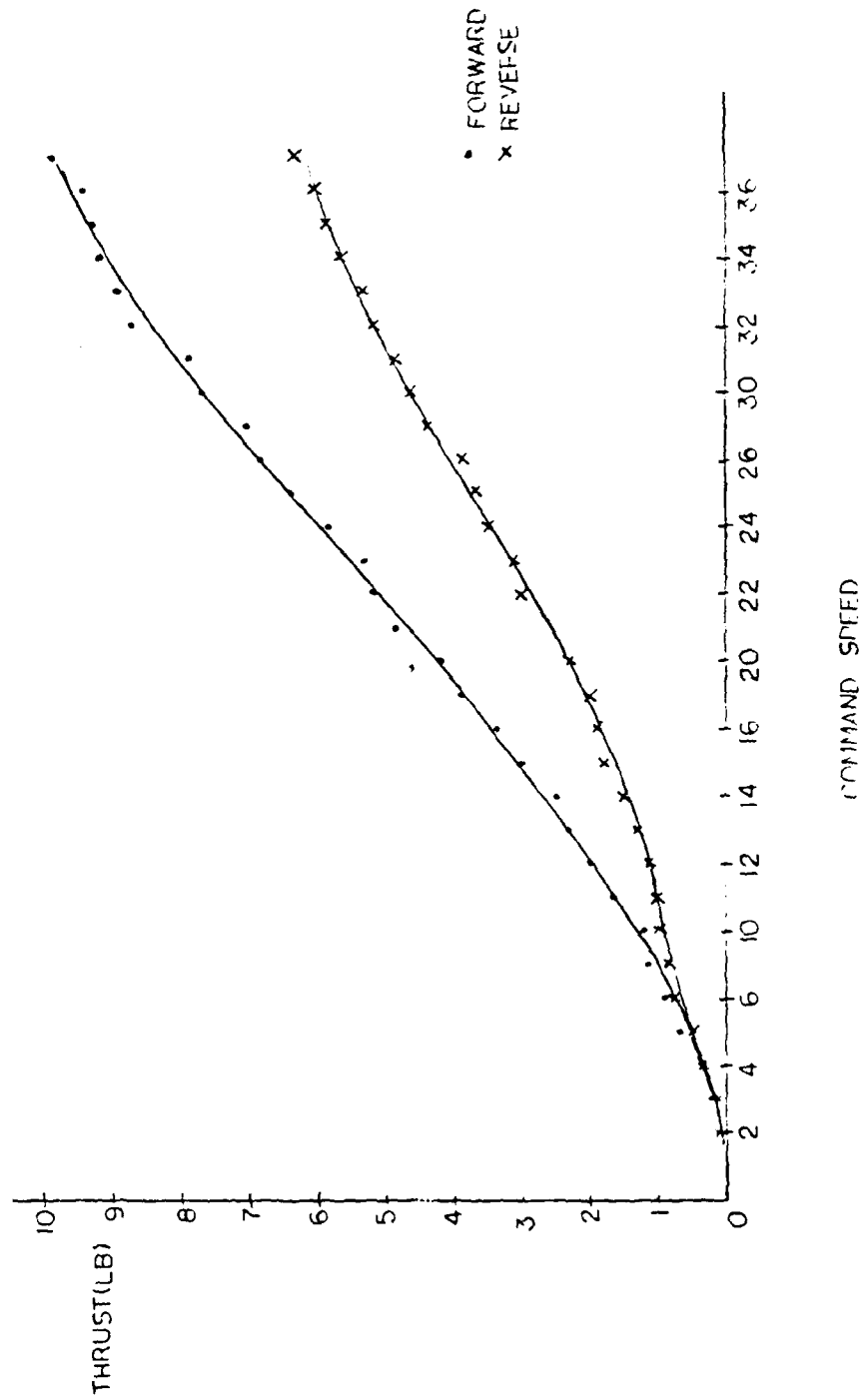
COMMAND SPEED

• FORWARD
x REVERSE

THRUST-VS-COMMAND SPEED
THRUSTER #2

FORWARD
REVERSE

THRUST(LB)

COMMAND SPEED

A-11

THRUST -VS- COMMAND SPEED   [RIGHT-HAND MINN-KOTA PROP]
UNMARKED THRUSTER

THRUST(LB)

. FORWARD
x REVERSE

COMMAND SPEED
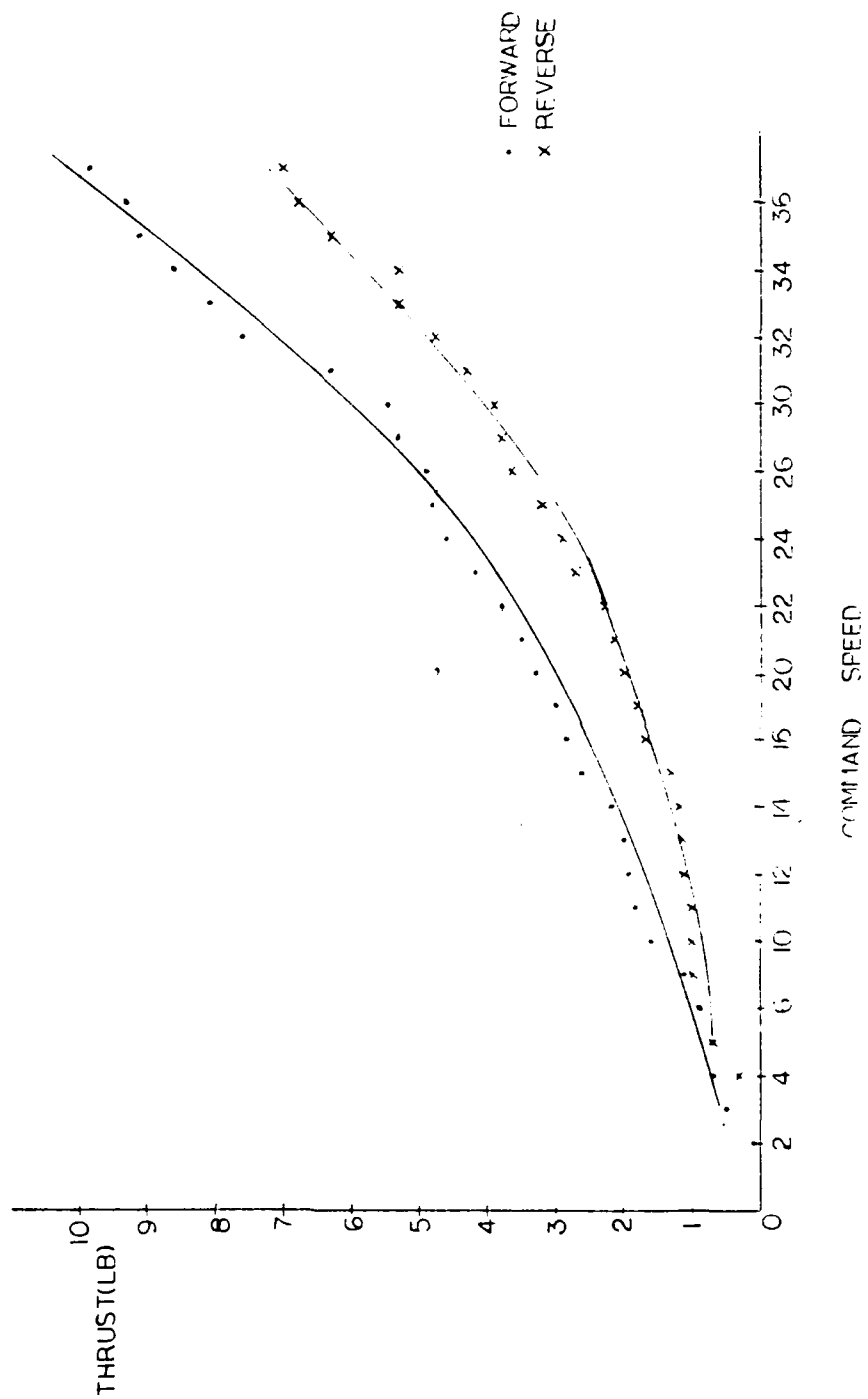
SIMS 68000 MONITOR

REV. 2.1

USER'S MANUAL

U.N.H. Marine Systems Engineering Laboratory
January, 1983
R.H. Lord

Revision History

1.0	6/8/81	R. H. Lord & D. J. Carroll
1.1	6/30/81	R. H. Lord
1.2	8/17/81	R. H. Lord
1.3	5/6/82	R. H. Lord
2.0	6/14/82	R. H. Lord
2.1	10/13/82	R. H. Lord
2.1V	11/16/82	R. H. Lord

1.0   Basic System Monitor

1.1   Hex load on UART #3

1.2   Buffered interrupt driven I/O

1.3   Transparent mode on all UARTS - removed interrupt I/O

2.0   24 bit address entry - binary load

2.1   Correction to binary loader

2.1V Embedded video monitor

END

CMME

CTC
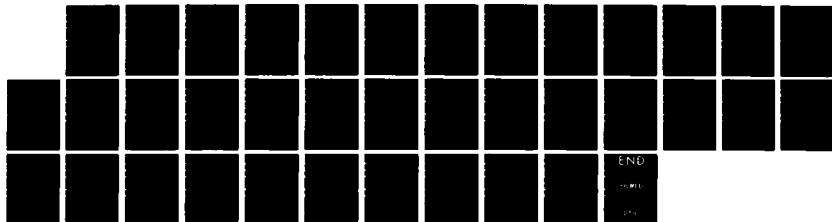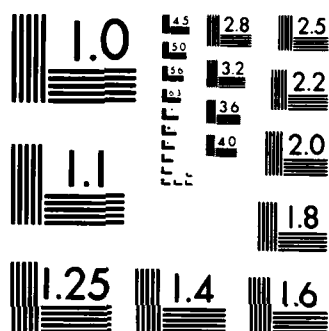
MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

FOREWORD

The 68000 micro-computer is an extremely versatile machine with a number of powerful features not normally available in micro-processors. Particularly useful are its TRACE mode and a number of explicit error traps. The address space directly available to the 68000 is 64 Megabytes (16M in each of four operational modes). An attempt to take advantage of all features of the machine would result in a monitor far larger than the 4K bytes we have alloted to the SIMS computer. This monitor is an attempt to preserve those features which we believe will be most useful SIMS development.

Features of this monitor include explicit error messages for all error traps, built-in random pattern memory testing, ability to breakpoint in ROM and a number of useful commands and user-available I/O subroutines. Excluded by this monitor are hex load to addresses above 65Kbytes (16 bit addresses) and use of bus-vectoring of interrupts.

The operation of this monitor may be a little different than the ones previously in use. Specifically, the syntax of the commands may not be familiar to those who have not cut their teeth on the 6800/6809. The commands are not buffered and scanned, but are entered directly. If a command calls for an address it should be entered as six hexadecimal digits. Leading zero's must be included (i.e. 00002E instead of 2E or 02E). If an error is made in entering the command it cannot be corrected with the back-space key. The command may be aborted by typing a non-hex character in an address or data field. This type of command processing was chosen because it was much simpler and more space efficient then a buffered command scanner. Most of the commands can be easily re-typed. If this method of command processing proves too error prone, a larger buffered input scheme can be substituted in a subsequent revision if space is made available for it.

SUMMARY OF MONITOR COMMANDS

B aaaaaa	–	Set breakpoint at "aaaaaa" (RAM or ROM)
B K	–	Clears all breakpoints
C	–	Continue after a breakpoint
D aaaaaa bbbbbb	–	Display a block of memory
G aaaaaa	–	Go jump to code at aaaaaa
H B	–	Binary load tape on port #3
H L	–	Load a HEX tape on port #3 (see L)
H P aaaaaa bbbbbb	–	Write HEX dump of aaaaaa-bbbbbb (see P)
I	–	Enable interrupts
L	–	Load a Motorola "S1" format HEX tape – lower 65K only
M aaaaaa	–	Examine/change memory at aaaaaa
O	–	Enter transparent mode with UARTS 0-3
P aaaaaa bbbbbb	–	Write Motorola "S1" tape of aaaaaa-bbbbbb
R	–	Display D0-D7 and A0-A7 from last break
S aaaaaa bbbbbb dd	–	Set "dd" into block of memory aaaaaa-bbbbbb
T aaaaaa bbbbbb tt	–	Set TRACE if PC is in aaaaaa-bbbbbb block
U	–	Turn off TRACE
X aaaaaa bbbbbb cccccc	–	Block move aaaaaa-bbbbbb to cccccc-
? aaaaaa bbbbbb	–	Random pattern memory test of aaaaaa-bbbbbb

----

(ctrl C) Restarts monitor
(ctrl S) Stops output
(ctrl Q) Re-starts output

B-4

DETAILED COMMAND DESCRIPTIONS

B aaaaaa   -   BREAKPOINT

The BREAKPOINT command displays a list of all breakpoints
and allows you to add a new address. Entering a 'K' will clear
all the entries. Any other non-hex character will return control
to the monitor.
On reaching the breakpoint, the processor will be halted and
the following message will be displayed:

                  BREAKPOINT AT aaaaaaa dddd cccc

This is followed by a dump of D0-D7 and A0-A7. The 'aaaaaaaa' is
the address of the breakpoint, the 'dddd' is the op-code at the
break and the 'cccc' is the current status register. The monitor
then asks if you wish to continue. You may do so at this time or
may exit to the monitor.   After changing memory or whatever, you
may continue from the breakpoint by typing 'C'.   The breakpoint
is not removed when it is encountered.
You may enter up to 30 breakpoints at a time. However,
doing so will slow the execution speed down.   The breakpoint is
implemented by turning on the 68000's TRACE flag and checking the
current program counter value against the breakpoint list. Since
this is done for each breakpoint address at the end of each
instruction cycle, this form of breakpoint will slow down
execution of the program by at least an order of magnitude. The
advantage of this method is that breakpoints may be set in ROM as
well as RAM.   If speed is critical to the execution of your
program then set a TRAP instruction at the desired RAM location
instead of a breakpoint.   You may re-vector this trap to the
breakpoint handler if you wish (see SETTING VECTORS).

C   -   CONTINUE - Restart from breakpoint (see above)

D aaaaaa bbbbbb   DISPLAY MEMORY

This command causes the block of memory from 'aaaaaa' to
'bbbbbb' to be displayed as an address followed by 16 hex bytes
followed by their ASCII equivalent value if any.   An integral
number of 16 location lines is always displayed.   Note that
'bbbbbb' may optionally be a count of locations if its value is
less than that of 'aaaaaa'. Thus 'D00 2000 00001F' would display
32 locations.

G aaaaaa   -   GO to location 'aaaaaa'

Loads the value '00aaaaaa' into the program counter, loads
the registers D0-D7 and A0-A7 from a buffer in the monitor RAM,
and starts the processor. The registers are loaded with the
value displayed by the 'R' command which are normally the values
that were in these registers the last time the processor was
halted by a breakpoint or trap.   The contents of this buffer may
be changed prior to executing the GO command if you wish to set a
register (see SETTING REGISTERS).

B-5

HB   -   Host Binary Load

     Load binary image through port #3 using ERG header block
format. A load information block is stored in 001240 - 00125C.

I   -   Turn on Interrupts

     Lowers interrupt mask to 0 in 68000 status registers,
enabling all interrupts.

L   -   LOAD Motorola HEX tape on Console port
H L -   LOAD tape on Port #3

     These two commands cause the monitor to enter the tape-load
mode. The character echo is turned off and input is ignored
unless it has the form 'Sldddd' or 'S9'. Bad characters and bad
checksums are noted if they occur. This mode can be terminated
only by receiving an 'S9'. The hex loader is limited to memory
locations below 010000 hex.

M aaaaaa   MEMORY examine/change

     The contents of location 'aaaa' are displayed as two hex
digits. If you enter two new hex digits, the contents of that
location will be changed and the next location displayed. If you
enter an up-arrow the previous location is displayed. If you
enter a return control will return to the monitor. If you enter
any other non-hex character, the next location will be displayed.

O   Set up transparent communication with UARTS 0 - 3

     The 'O' command prompts th ooperator for a UART #. The
number entered is interpreted module H and used to select UART #0
- 3. The monitor then enters transparent mode in which all
characters received by the selected UART are sent to the console
and vice-versa. Exit from this mode is by a ctrl-A from the
console (the only character that connot be sent to the external
UART).

P aaaaaa bbbbbb   'PUNCH' Motorola S1 format HEX tape on console
H Paaaaaa bbbbbb   Same as above on Port 3

     Write the data block 'aaaa-bbbb' out to tape in the Motorola
format of 'Slxxxx' followed by terminating 'S9'. In the console
mode (P) a ten second wait is inserted before the dump begins and
again after it is completed. This allows you time to connect and
disconnect the recorder. In the Port 3 mode (H P) the dump is
immediate.Dump is limited to locations below hex 010000.

R   -   Display REGISTERS

     This command displays the 32-bit contents of the data and
address registers D0-D7 and A0-A7 at the time of the last
breakpoint or (ctrl C) exit. These values are re-loaded at the
time a Continue or Go command is executed.

S aaaaaa bbbbbb dd   SET Memory Block

The block of memory from 'aaaaaa' to 'bbbbbb' is filled with
the data 'dd'. This command is useful for initializing tables.

T aaaaaa bbbbbb ff   Set TRACE Mode

Actually the current value of 'aaaaaa bbbbb ff' appear as
soon as you type the 'T' but you may re-type any or all three of
these values.   To accept the current value for any field, type a
space instead of a HEX number.
This command places the 68000 in the TRACE mode.   When the
trace exception occurs, the program counter is checked to see if
it is between 'aaaaaa' and 'bbbbbb'.   If it is and if the flag
byte 'ff' is non-zero then the trace is performed and

          @ aaaaaaaa cccc ssss

is displayed on the console.   The 'aaaaaaaa' is the current
address being traced.   The 'cccc' is the op-code at that address
and the 'ssss' is the status register.   In addition, if the flag
byte 'ff' is greater than 7F hex (127 dec.) then a full display
of the data and address registers also appears.

U   -   UNTRACE

Turns off the TRACE mode display by setting the trace flag
byte to zero (see TRACE above).   Checks to see if there are any
breakpoints.   If not, the trace bit of the 68000 status register
is also cleared.

X aaaaaa bbbbbb cccc   BLOCK MOVE

Copy the data block 'aaaaaa - bbbbbb' to the address block
from 'cccccc -' to C+(B-A).

? aaaaaa bbbbbb   Test Memory

Executes a random pattern test of the memory block between
'aaaaaa' and 'bbbbbb'.   This test writes random numbers in each
memory location.   It then re-seeds the random number generator
and checks each memory location.   If all locations check then an
exclamation mark is printed and a new seed used for the next
pattern cycle.   If an error is encountered the address, intended
data and real data are displayed.

DETAILED USER SUBROUTINE DESCRIPTION

READCH 0F80

Input an ASCII character to D0.L using the input routine pointed to by INVEC at $1000. On power-up this is normally set to GETCHR (at $01A0) and changed to CIN (at $0E00) when in the interrupt mode. This routine masks the character to 7 bits and echo's it through OUTCH if the echo flag ($1220) is non zero.

OUTCH 0F84

Outputs the contents of the D0.B as an ASCII character using the output routine pointed to by OUTVEC at $1004. On power-up this normally is set to BRKCHR (at $01B2) where it tests the keyboard input for a break (ctrl C) or stop (ctrl S). This vector is changed to COUT (at $0E36) when the interrupt mode is enabled. The contents of D0 are unaffected by these two routines.

OUTS 0F8C

Outputs a space (ASCII $20) through OUTCH. No registers are disturbed by this subroutine call.

OUTCR 0F90

Outputs a carriage return - line feed pair (ASCII $0D, $0A) through OUTCH. No registers are affected by this subroutine.

PSTRNG 0F94

Output an ASCII character string through OUTCH. This subroutine assumes that A6 points to the first character of the string and that the string ends with a null character ($00). The subroutine ends with D0.B containing the null and with A6 pointing to the next location after the null.

OUTHX8S 0F98

Outputs the contents of D0.L as 8 hexadecimal digits followed by a space using OUTCH. The present implementation alters the contents of D0.B and adds each digit to the checksum (in $1099) for the PUNCH routine.

OUTHX4S 0F9C

Outputs the contents of D0.W as 4 hexadecimal digits and a space. See OUTHX8S above.

OUTHX2S 0FA0

Outputs D0.B as 2 hexadecimal digits and a space. See above.

INHEX4 0FA4

Input four hex digits via READCH and assemble them into a 16
bit word in D0.L. If any character is non-hexadecimal this
routine sets the carry and returns the non-hex character in D0.B
instead of the hex word. If the characters are all good it adds
them to the checksum (in $1099) for the LOAD routine and then
returns with the assembled hex number in D0 and with the carry
cleared.

INHEX2 0FA8

Input two hex digits to D0.B. See INHEX4 for details.

LIMITS 0FAC

Input two 6-digit hexadecimal numbers to A0 and A1. This
subroutine is useful for setting up address limits. It first
calls INHEX4 and places the hexadecimal word in A0.L. It then
outputs a space via OUTCH and calls INHEX6 again. If this number
is of larger magnitude than the first it is placed in A1 and the
difference is calculated and placed in D0.L. If the second is of
smaller magnitude than the first, then it is placed in D0.L and
then summed with the first number and the sum placed in A1.L.
Thus you may enter either two addresses or a start address and a
count and the routine will return with two addresses in A0, A1
and a count in D0. Returns with carry clear if all entries were
valid hex digits. If a non-hex character is encountered the
routine returns with the carry set and the character in D0.B. A0
will be correctly set if the error was in the second entry.

HEXDEC 0FB0

Convert D0.L from binary to a signed decimal string. Enter
with A6 pointing to top of string buffer. This routine places a
null at the end of the string and then moves A6 backward as each
succesive ASCII digit is generated. If the number is negative,
the ASCII minus sign is then placed at the beginning of the
string. The routine exits with A6 pointing to the first
character of the string and with the original binary number
intact in D0.L. This routine is limited to values between
-655,359 and +655,359. Numbers larger than this will cause
overflow in the divide routine.

HXDOUT 0FB4

Outputs D0.L as a signed decimal number to the OUTCH
routine. This routine creates a buffer in $1200 - $1214, and
calls HEXDEC followed by PSTRNG. The number is left justified.
Both D0 and A6 are altered by this routine.

WAIT 0FB8

Wait D0.W tenths of a second. Enter with D0.W containing
the desired delay in tenths of a second. This routine presently

operates by software timing loops. Its accuracy is a function of
the access timing for both monitor ROM and stack RAM. Any
interrupt handling time is also added to the delay. Useful for
non-critical timing applications only. DO.W is destroyed.

RANDOM OFBC

        Returns a pseudo-random number in DO.B each time the routine
is called. The seed is in the word stored at $109A. This can be
re-seeded to repeat an identical sequence.   D1 is set to zero by
this routine as presently implemented.

COLD OFDO

        Entry point for re-initializing monitor. Resets all vectors
to their initial power-up values. This is equivalent to hardware
reset. Turns off interrupt I/O.

MON OFD4

        Return to monitor command prompt.   Leaves system configured
as it was prior to entry.   Does not store register contents in
temporary, so "R" command does not display this information.

                        SETTING SYSTEM VECTORS

        The Input/Output routines and 68000 traps are all indirectly
vectored through system RAM in locations $1000 - $108F.   These
are shown in Table 1.   These locations each contain a 32-bit
address pointing to the routine which will perform the vectored
function.   On power-up, the I/O vectors are set to GETCHR and
BRKCHR and the system error traps are set to individual error
handlers.   The interrupts and software trap vectors are set to a
trap handler within the monitor which prints an error message.

        The user will probably want to re-direct many of these
vectors to other handling routines.   This may be accomplished by
storing the new vector address as a long word in the appropriate
vector location. For temporary testing this can be done with the
"M" command in the monitor for all vectors except INVEC and
OUTVEC.   These can't be modified while the monitor is using them
to input the new addresses.   A much more satisfactory method of
changing vectors is to include the change in your program
initialization routines.   This allows you to recover if you need
to press RESET which would destroy all your vectors. The address
can be set with a long word move instruction such as:

        MOVE.L #GETCHR, INVEC

SETTING 68000 REGISTERS

When the GO command is executed the monitor loads D0-D7 and
A0-A6 from a table in RAM before jumping to the GO address. This
table receives the contents of these registers whenever a trace,
breakpoint or error trap occurs but the contents are random on
power-up. The user may directly modify this table to initialize
any of the registers (except A7) before jumping to a program.
The table is in memory locations $109C through $10DB with D0 in
$109C through $109F and the other registers in consecutive double
word locations. See Table 1.

,

SUMMARY OF MONITOR COMMANDS

B aaaaaa   - Set breakpoint at "aaaaaa" (RAM or ROM)
B K   - Clears all breakpoints

C   - Continue after a breakpoint

D aaaaaa bbbbbb   - Display a block of memory

G aaaaaa   - Go jump to code at aaaaaa

HB   - Binary load tape on port #3
H L   - Load a HEX tape on port #3 (see L)
H P aaaaaa bbbbbb   - Write HEX dump of aaaaaa-bbbbbb (see P)

I   - Enable interrupts

L   - Load a Motorola "S1" format HEX tape -
                      lower 65K only

M aaaaaa   - Examine/change memory at aaaaaa

O   - Enter transparent mode with UARTS 0-3

P aaaaaa bbbbbb   - Write Motorola "S1" tape of aaaaaa-bbbbbb

R   - Display D0-D7 and A0-A7 from last break

S aaaaaa bbbbbb dd - Set "dd" into block of memory aaaaaa-bbbbbb

T aaaaaa bbbbbb tt - Set TRACE if PC is in aaaaaa-bbbbbb block
U   - Turn off TRACE

X aaaaaa bbbbbb
  cccccc   - Block move aaaaaa-bbbbbb to cccccc-

? aaaaaa bbbbbb   - Random pattern memory test of aaaaaa-bbbbbb

----

(ctrl C) Restarts monitor
(ctrl S) Stops output
(ctrl Q) Re-starts output

M68000-Based Navigation Computer for EAVE-East

Rationale

The IM6100-based navigation computer has some shortcomings, which make it unacceptable for future technology development on the vehicle. For example:

- 12-bit counters limit the effective range of the system to approx. 200 feet.

- An auxiliary math processor is required to compute the trigonometric functions. It has high power consumption, and requires elaborate strobing circuitry to minimize battery drain.

- Assembly-language programming makes software changes difficult to find, and implement.

- Extended features, such as multi-beam listening, and autocalibration of beacon location, are impossible.

Design Philosophy

To overcome these problems, an M68000-based navigation computer has been designed for EAVE-East. The reasons for selecting the M68000-based design are manifold:

- An M68000 is used for the EAVE-East command computer. The development stations, hardware, and software expertise exist in-house.

- 16-bit architecture allows the range to be extended to 800 feet (total path travel, assuming no delays) with better than 3/10-inch resolution. Under software control

the _theoretical_ range is 10,000 miles with 1,600 feet
resolution.

- The M68000 has enough computational power to:

    - eliminate the math processor.

    - perform autocalibration of transponder location.

    - listen to several beacons for long-range navigation.

    - handle inputs from future navigational, and control
      options (i.e. compass, IMU's, orientation sensors).

- Programming can be done in a high-level language ("C"),
  making it easier to modify programs.

In addition, most of the analog electronics were updated to
provide higher pinger output power, and better noise performance.

Configuration

The structure of the navigation computer is shown in Figure
1. The digital portion of the computer consists of the CPU card,
a UART card, and one or more 32Kbyte memory cards.

The basic navigation algorithm should easily consume less
than 32KB. The autocalibration algorithm will need approximately
32KB by itself. A slot for future expansion will be included in
the card cage.

The analog section consists of a receiver preamp/transmitter
power boost (PRA/PB) located at each transducer, a receiver (3-
channel), three mechanical filters (110, 114, 118KHz) per
channel, and a nine-channel detector card.

The detectors produce a TTL-compatible, 1ms pulse when a
transponder return pulses arrive at each transducer. These
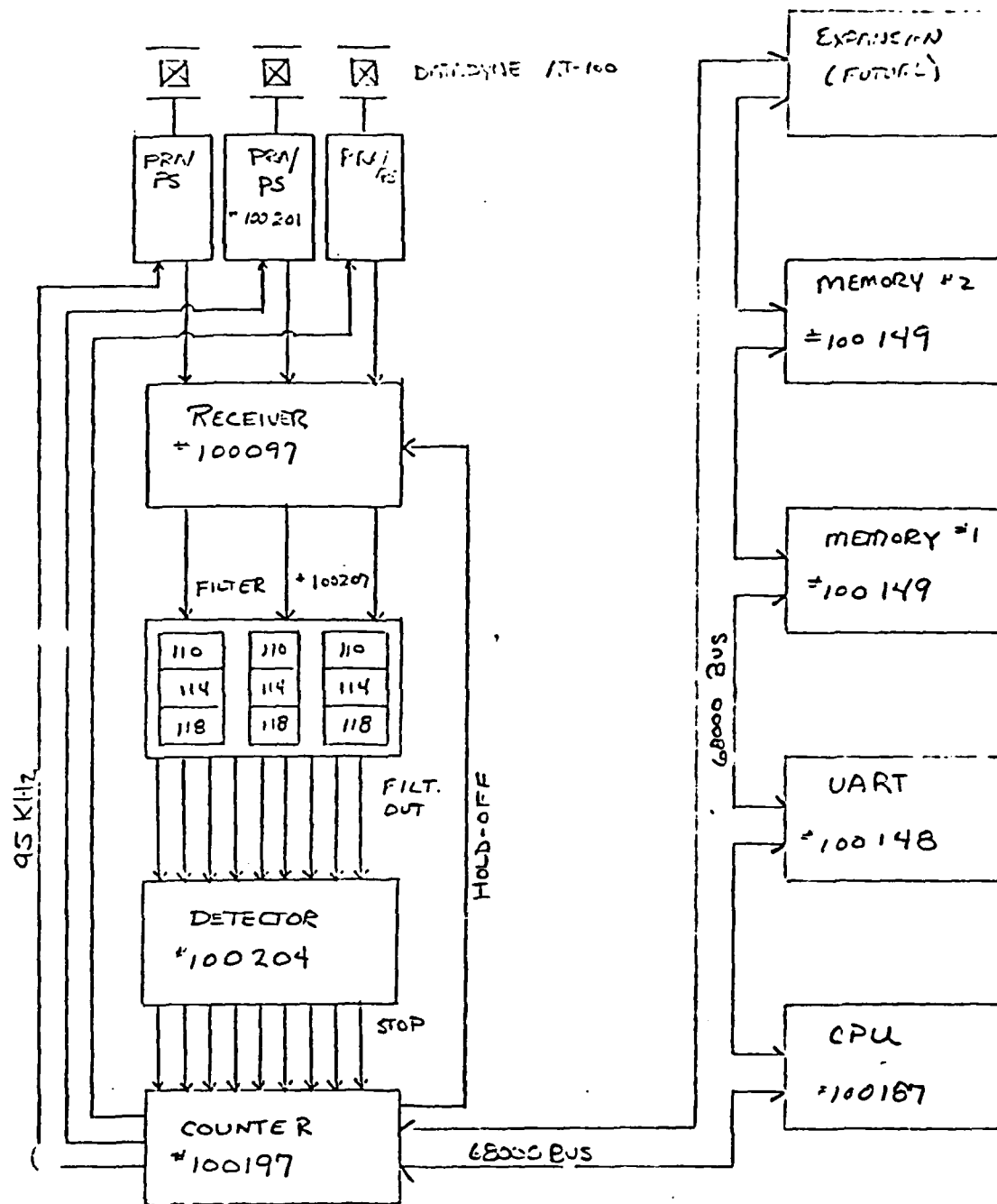pulses are used to stop each of the nine counters on the counter
board.

Figure 1 - 68000 Navigation Computer Block Diagram

TABLE 1

AIMS NAV COUNTER MEMORY

ADDRESS MAP

```
003001 -   Illegal Address
003003 -   Illegal Address
003005 -   "N-Clock" Counter LSB
003007 -   "Hold-Off" Counter LSB
003009 -   Interrupt Status Reg U6 (Read only)
00300B -   Interrupt Status Reg U6 (Read only)
00300D -   "N-Clock" Counter MSB
00300F -   "Hold-Off" Counter MSB
```

```
003011 -   Illegal Address
003013 -   Illegal Address
003015 -   Counter 1  LSB
003017 -   Counter 2  LSB
003019 -   Interrupt Status Reg U7 (Read only)
00301B -   Interrupt Status Reg U7 (Read only)
00301D -   Counter 1 MSB
00301F -   Counter 2 MSB
```

```
003021 -   Illegal Address
003023 -   Illegal Address
003025 -   Counter 3 LSB
003027 -   Counter 4 LSB
003029 -   Interrupt Status Reg U8 (Read only)
00302B -   Interrupt Status Reg U8 (Read only)
00302D -   Counter 3 MSB
00302F -   Counter 4 MSB
```

```
003031 -   Illegal Address
003033 -   Illegal Address
003035 -   Counter 5 LSB
003037 -   Counter 6 LSB
003039 -   Interrupt Status Reg U9 (Read only)
00303B -   Interrupt Status Reg U9 (Read only)
00303D -   Counter 5 MSB
00303F -   Counter 6 MSB
```

```
003041 -   Illegal Address
003043 -   Illegal Address
003045 -   Counter 7 LSB
003047 -   Counter 8 LSB
003049 -   Interrupt Status Reg U10 (Read only)
00304B -   Interrupt Status Reg U10 (Read only)
00304D -   Counter 7 MSB
00304F -   Counter 8 MSB
```

TABLE 1 continued

```
003051 - Illegal Address
003053 - Illegal Address
003055 - Counter 9 LSB
003057 - Counter 10 LSB (not used)
003059 - Interrupt Status Reg U11 (Read only)
00305B - Interrupt Status Reg U11 (Read only)
00305D - Counter 9 MSB
00305F - Counter 10 MSB (not used)
```

```
003061 - Master Stop
003063 - Master Stop
003065 - Master Stop
003067 - Master Stop
003069 - Master Stop
00306B - Master Stop
00306D - Master Stop
00306F - Master Stop
```

```
003071 - Start
003073 - Start
003075 - Start
003077 - Start
003079 - Start
00307B - Start
00307D - Start
00307F - Start
```

Notes: 1. Even Addresses will not give VPA.

2. Top 2 bytes of Address may vary from

OOOOXX - OOFFXX
OO3OXX selected arbitrarily.

Counter Board Operation

The counter board serves as the analog/digital converter for the system. The counter board appears as an M68000 peripheral to the CPU, (i.e., the processor must assert VMA, look for VPA, and synchronize data transfers with the E-line. The E-line is also used as the counter clock (400KHz).

The counters used are the RCA CDP1878 CMOS dual counters. There are six chips for a total of twelve counter sections. One section is used as a divide-by-N counter for the E-signal. N is chosen to provide the required transponder range. Another counter is used for "hold-off", and provides a programmable-width pulse to disable the receiver, and counter circuitry for a few microseconds after the pinter has been triggered.

The counter board also produces the 95KHz burst used to ping and circuitry used to route the burst to the correct pinger.

A transponder cycle begins by jam-loading the range number (N) into the first counter, then the hold-off time into "hold-off" counter. The appropriate pinger transducer is selected.

The number $FFFF is then jam-loaded into the nine counters (the last counter is disabled).

A start pulse is generated by writing to the appropriate memory location (see Table 1). Simultaneously a 1ms long, 95KHz burst is sent to the selected pinger, and the counters begin couting down.

After some arbitrary time period, a master stop location is addressed, and the counters are interrogated. Overflows are looked for, and the numbers are read from the counters. The nine times are then used to calculate x, y, and heading information for the command computer.

Status

The system is not complete at this time. The hardware design is complete, but the software has not been written, and much of the testing needs to be done.

68000 CPU - complete

UART - complete

Memory - complete

Counter - breadboarded, not tested

Preamplifier/Power Stage - complete

Filter - complete

Reciever - not built, but design proven/AGC need tweaking

Detector - built, not tested

Software - outlined

APPENDIX D

NAVIGATION SYSTEM SENSITIVITY

Discussion of Errors

In the course of this work several sources of error were located and their magnitude was approximated. The errors can be divided into two general categories: random and motion errors. The term error is used loosely as any unaccounted for or nonlinear phenomenon. So, in our case motion errors, though deterministic, are not compensated for in the algorithm and lead to incorrect predictions of position.

Random errors exist in the system due to uncertainty in the delays of all transponders and hydrophones. The local sound velocity error is also considered to be random error for our purposes. The above parameters will act as "noise" on all of the range measurements so their effect must be considered in calculations. The speed of sound error is considered a random because of its complex dependence on temperature, pressure, and density of the water. It is not uncommon for the sound velocity to differ by .5% locally. A .5% error in sound velocity leads to a .5% error in range measurement which is one foot at a range of 200 feet.

In many cases the error found denoted the worst case error (see Table 1). In order to get a more useful representation of errors one standard deviation was used (see Table 2). The worst case error corresponds to three standard deviations as a general rule so the statistical errors were divided by three. Going further the rms error was found by taking the sum of the squares and then the square root.

TABLE 1

ABSOLUTE ERRORS

Statistical

Transponder detect delay   .3± .1 ms

Turn around detect delay   10, 30, 50 ± .01 ms

Hydrophone detect delay   .55 ± .035 ms

Total delay means   10.85, 30.85, 50.85 ± .145 ms

Speed of sound = 4800 ± .5% ft/s at 60°F

Motion Errors

Errors in 2 way times due to skew

Errors in 2 way times due to rotation

Errors in 2 way times due to transponder delays

   (function of velocity, heading, heading dot, distance)

TABLE 2

Determination of Statistical Error

Hydrophone detect delay uncertainty .035 ms

Transponder turn around delay uncertainty   .01 ms

Transponder detect delay uncertainty   0.1 ms

The above numbers are absolute errors which are three standard deviations away; for rms error use 1 S.D.

$.035 \text{ ms} \div 3 = 1.1667 \times 10^{-5} \text{ sec}$

$.010 \text{ ms} \div 3 = 3.333 \times 10^{-6} \text{ sec}$

$.100 \text{ ms} \div 3 = 3.333 \times 10^{-5} \text{ sec}$

$\text{RMS STAT ERROR} = \text{SQRT} \quad (4800(1.1667 \times 10^{-5}))^2 + (4800(3.33 \times 10^{-6}))^2 +$

$(4800(3.33 \times 10^{-5}))^2 = .17 \text{ ft} = 2.04 \text{ inches}$

Motion Errors

It was found that motion errors are dependent on speed, rotational speed, skew and distance of the vehicle from transponders. A progam was written to simulate these errors as the vehicle moves from a source to destination.

The maximum vehicle underwater speed is about one Know. For a safety margin of 50% 1.5 Knots will be used in this analysis where the maximum motion error is found. At a range of 200 feet and a speed of 1.5 Knots, the vehicle will move

$$1.5 \text{ Knots} \times 1.688 \frac{ft/s}{Knots} \times ((200 \text{ ft.} \times 2)/(4800 \text{ ft/s}) + 50.5 \text{ ms})$$

$$= .339 \text{ ft.} = 4 \text{ in.} = 2 \text{ in.}, \text{ etc.}$$

This is an absolute worst case error at a given range.

The problem of rotation is considered next. The vehicle gains are set such that the rotation rate is nominally not larger than .157 rad/sec. Again at a range of 200 ft the Δt is

$$((200 \times 2)/4800 = .0508 = .134 \text{ sec} \quad .134\text{s} \times .157 \text{ rad/s} = .02104 \text{ rad}$$

In this amount of time the vehicle has gone through .02104 x 1.5 ft. = .0316 ft. of arc or .38 inches which is then resolved into x and y errors. Since the fastest rotation is .157 rad/s and the corresponding error is .38 inches it can be stated that the rotation of the vehicle has little effect on the range measurements and therefore negligible effect on the calculated position. The skew orientation of the vehicle does not present a problem if the data is interpreted properly. The difference in hydrophone position can change by as much as three feet from the unskewed to the worst case skew position.

In summary, the transponder and hydrophone delays account for a rms error of approximately 2.04 inches. The error due to motion of the vehicle in the water at 1.5 knots at various ranges is:

Speed	Range	Error
1.5 Knot	200 ft.	4 in.
1.5 Knot	100 ft.	2 in.
1.5 Knot	50 ft.	1 in.

The error due to a sound velocity error is:

Percent Error in Sound Velocity	Range	Error
.5%	200 ft.	1 ft.
.5%	100 ft.	6 in.
.5%	50 ft.	3 in.
.25%	200 ft.	6 in.
.25%	100 ft.	3 in.
.25%	50 ft.	1.5 in.

APPENDIX E

EAVE VEHICLE CAMERA

Introduction

The following paper is not meant to be either a detailed report or a technical manual. It is simply an account of how we procceded in designing and fabricating the project that was assigned.

The project assigned was to work in conjunction with the Marine Systems Division at the University of New Hampshire. During our introductory meeting with the members of this group we discussed the possibility of designing an automated camera system that could be attached to an existing submersible vehicle. It was required that our system be able to take commands and send back messages to the micro-computer that controls the vehicle. The system would also have to be designed around an already purchased Sankyo EM 40-XL 8mm movie camera.

On the following page is an outline of what we thought would have to be done in order to complete the project.

E.A.V.E.

CAMERA SYSTEM

I. Research methods of underwater photography.

    - Lighting techniques
    - Focusing
    - Film types

II. Obtain all available information on the Sanko EM40XL movie camera.

    - Dimensions, schematics, operating instructions, ect.
    - Obtain information on light meters. Those that can be used with the light levels at the depth the camera will be working at.

III. Investigate the use of an alternative camera system.

    - For better picture quality
    - More efficient use of power

IV. Determine what automated controls will be required.

    - On/Off
    - Lighting
    - Frame rate, etc.

V. Research and design of camera housing. (Carl)

    - Materials to be used
    - Stress calculations

VI. Electrical interface design. (Jim)

    - Decide on I/O methods
    - Designing any hardware and software required

VII. Final assembly and test.

    - The final system should consist of a housing containing a camera with the electrical and mechanical components required to perform all the automated controls.

Mechanical Design

I started my design by determining just where I wanted
to mount the camera housing. The mounting location was very
important to the quality of the photographs as I explained
on pages 16 and 17 of my note book. My next step was to deter-
mine the method I would use to mount the camera and electron-
ics within the housing. Easy acessibility to the components,
as well as being free from disturbance caused by deflection
in the housing that will occur under deep sea pressures was
of prime importance here. The next step was to decide how I
was going to mount the housing onto the vehicle. The limiting
factor here was that the housing had to have two degrees of
freedom: left & right, up & down. it also had to be easily
moveable from one side of the vehicle to another. This was
to give complete freedom as to where they could photograph
with respect to the vehicle: all four sides as well as a
wide range of area on any of the four sides. Now that this
was all determined I had a good handle on just what I needed
for a housing configuration.

My next step was to determine what materials I was going
to use. I choose 6061-T6 aluminum for four reasons:
1) Its resistance to corrosion.
2) It was a stock material for aluminum tubing.
3) I needed a non-magnetic material because of the
vehicles navigation system.
4) It had a high yield strength for its price.
I then choose a high tensil strength clear plastic for the
window plate. I choose this over glass mainly because of
cost, as well as its resistance to breaking..

I then did all the necessary stress calculations required on the components. I used two different methods for these calculations.to double check each other. Also the second method I used was much more detailed and accurate than the first method. These calculations are on pages 34 thru 43 in my note book. I then detailed all my components, ordered the materials and spent the rest of the semester fabricating the components.

Electrical Design

Before any of the electrical design could begin, two things had to be known. They were, what controls would be required to do the job and what the camera's abilities were.

After numerous meetings with our technical advisors at Marine Systems, it was decided that there were five functions that they would like to be controllable; 1) camera on/off, 2) variable frame rates, 3) a frame counter, 4) presettable frame count and 5) a light meter to measure ambient light in the environment.

Knowing what had to be controlled, it then had to be determined what the camera was capable of doing. I first contacted Sankyo of America, but after two phone conversations with them it was very apparent that they were not geared to answer technical questions as to the performance of their cameras. So it was required that I test the camera myself to find its limitations. It was found that using the remote control jack to advance frames, the camera would only work reliably at speeds up to 4-5 frames per second, any faster and the switching became very erratic. The Marine Systems group had originally wanted to run the camera externally at up to standard movie rates of 12 and 18 frames per second, but we now knew that this would be impossible without modifications to the camera's electronics. They decided they did not want to modify the camera in that way,

so it was decided that rates up to 4 frames per second would be fast enough. I also contacted a number of photo-dealers in order to find out how the camera would perform in very low light situations, as it would encounter under-water. What we found out was that the camera we had was advertised as one that was good in low light applications. What was meant by this, though, was that the camera could be used in normal room light without added illumination. One of the biggest limititions was that the fastest film available was only ASA 160. What all this meant was that in many of its applications our camera system would require an additional light source, be it constant or strobe. It was also found in the tests that the camera drew more cur-rent while in operation then previously thought; 20 ma at idle, approximately 300 ma at 18 frames per second and instantanious currents of over 500 ma in its single frame mode. This required that the camera would have to be tied directly to the batteries that power the vehicle and not off the power to the electronics.

Knowing the controls required and the limitations of the camera, the final design could begin. Figure 1 shows a basic block diagram of the result. This controller recieves and transmits information as an 8 bit digital word. This word is decoded to do the various functions. The three most significant bits determine what function will actually be done. The lower five bits sent are used to program the

frame rate, to preset the number of frames to be shot and to turn the camera on, depending on what function was selected.

Figure 2 is a detailed schematic, (minus UART controls). The following is a short description of its operations.

The frame rate is determined by the output frequency of the 4047 astable. One cycle per second equals one frame per second, two cycles per second equals two frames per second, etc., up to four frames per second. The different frequencies of operation are obtained by switching in different R-C time constants to the 4047. This is accomplished by switching in different values of resistance using a 4051 One of eight switch. The three bit word used to select the channels in the 4051 is latched from the outputs of the UART by a 4174 Hex-D-flip flop.

The counter circuitry is made up of two cascaded 4040 twelve stage binary counters. The count is read back as two eight bit words, giving a maximum count of 2^{16} frames,. the count is held for transfer by two 74LS244 tri-state latches.

This camera system can be pragrammed to shoot at a specified frame rate indefinately, or it can be preset to shoot a specific number of frames. The latter is done with two 4029 synchronous presettable down-counters. Each counter is loaded individually by latching a four bit word into its inputs. The counters are cascaded so that up to 2^8 or 256 frames can be preset.

The light meter is built up around an ADC 1211 A/D
converter. The A/D recieves a analog signal from a vol-
tage divider network consisting of a CDS light sensor and
a fixed resistence. The A/D is configured so that it is
continually sampling and latching in the latest light
reading. The data is stored in two 74LS374 octal D-flip-
flops. The light intensity is read back as two eight bit
words.

Conclusion

In conclusion we would like to point out that our system meets all performance criteria. All the components fit together and the electrical controller performs all the necessary automated functions.

We completed what we set out to do. The Marine Systems Lab is happy with the product, and we did keep pace with our estimated time schedule.
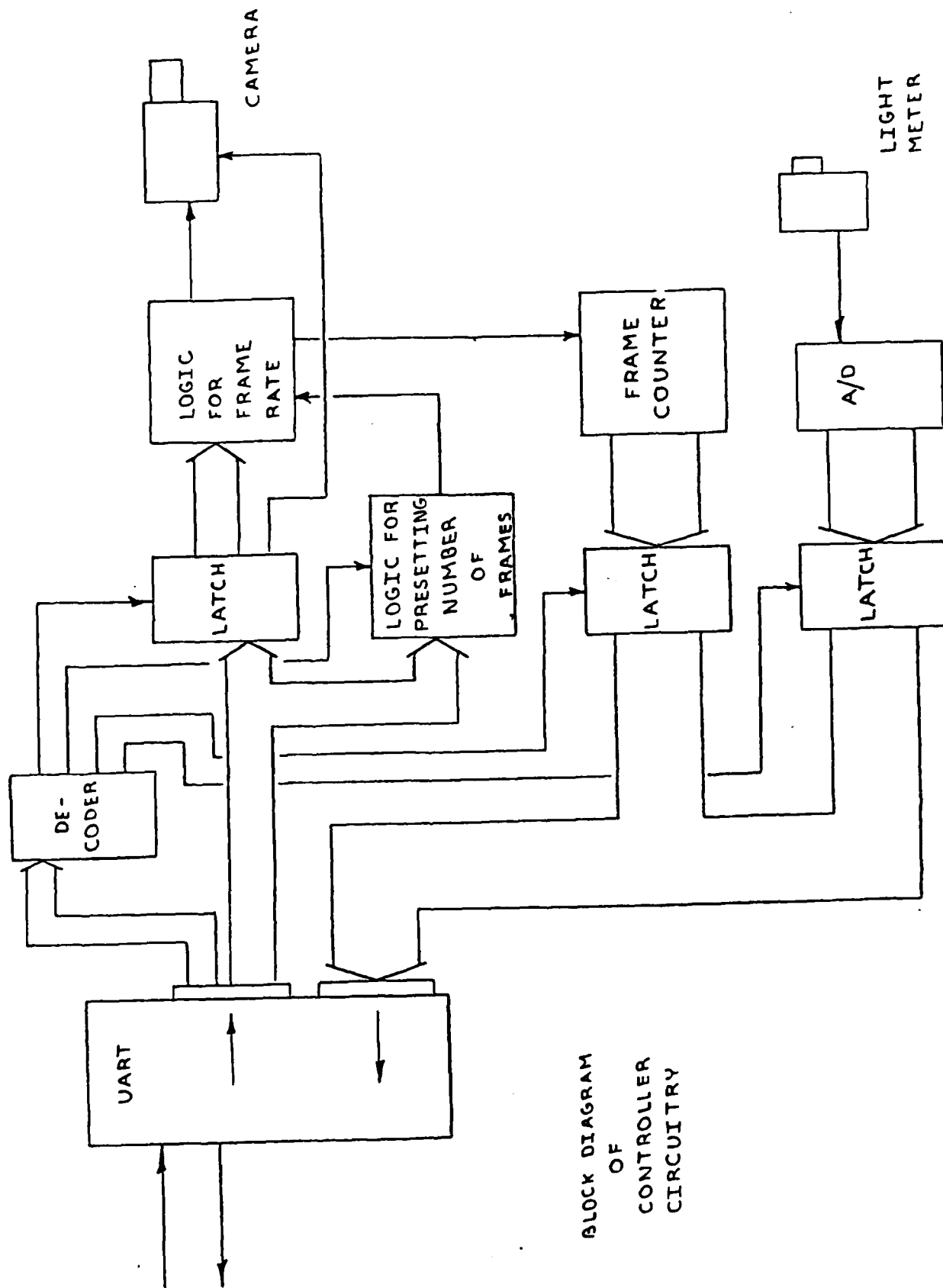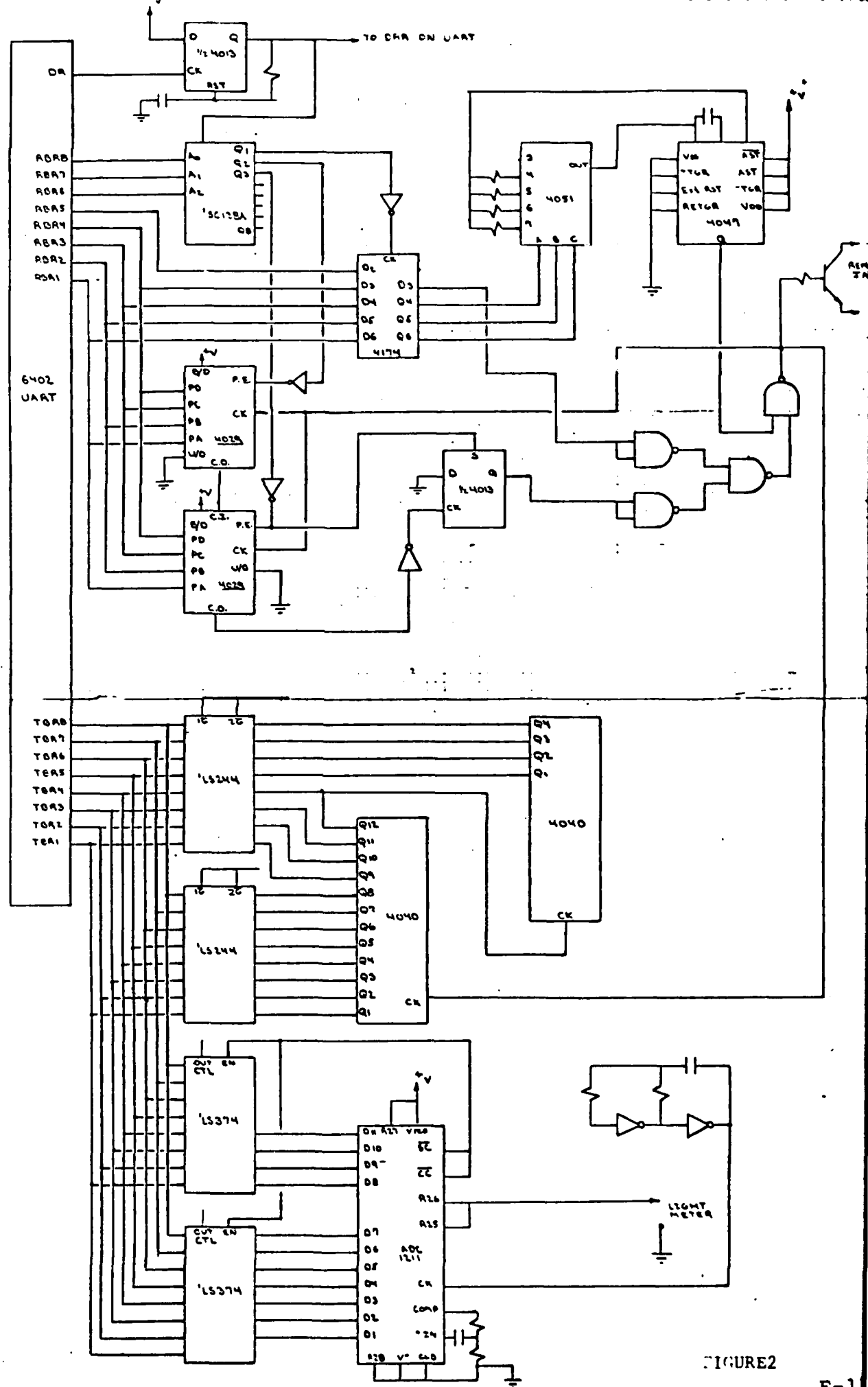
BLOCK DIAGRAM
OF
CONTROLLER
CIRCUITRY

FIGURE 1

FIGURE2

E-1

APPENDIX F

MECHANICAL DRAWINGS FOR EAVE

<u>Thrusters</u>:

        100021C - Thruster assembly
        100022A - Pressure well face cap
        100023B - Diaphram housing
        100025B - Casing length
        100026A - Propeller shaft
        100052A - Propeller bearing shaft
        100056A - Shaft assembly
        100063A - Prop shaft bushings
        100066D - Thruster final assembly
        100067A - Diaphram plate
        100162C - Kort nozzle and thruster
        100163B - Kort nozzle mounts
        100164B - Shaft housing modifications
        100165B - Motor housing lower bulkhead
        100166B - Motor housing upper bulkhead
        100167B - Motor housing
        100168B - Bellows retaining disk
        100169B - Motor modifications
        100170C - Motor housing assembly
        100183B - Thruster physical layout

<u>Navigation</u>:

        100140A - Navigation transducer frame (sketch)
        100171B - Transducer mount

<u>AIMS Sonar</u>:

        100178C - AIMS echo sounder pressure case
        100179C - AIMS echo sounder lower end cap

<u>Frame</u>:

        100211C - EAVE dimensional outline
        100191B - Thruster mount parts
        100192B - Thruster mount assembly
        100182D - Battery frame

<u>Computers</u>:

        100019 - Side rail computers
        100083B - Face cap assembly
        100070 - Edge connector (side rails)
        100111B - Thruster card cage side rails
        100137B - Bulkhead for bubble memory
        100139B - EAVE thruster card cage
        100142B - 68K card cage and mounting rails
        100152B - EAVE control end cap
        100173D - Thruster heat sink layout

APPENDIX G

EAVE SYSTEM SCHEMATICS LIST

Bus Layouts:

    100058A - Thruster computer bus
    100116B - Navigation receiver bus
    100189B - MBM Bus
    100147B - 6100 CPU slotted bus
    100186D - 68000 bus (see split 68000 CPU)

6100 Computer Systems:

    100015D - 2K Prom
    100042D - 3K Prom
    100043D - MEX/Drvr.
    100044D - Bus driver
    100045C - 6100 CPU
    100046C - UART
    100047D - 4K RAM

    100051C - Thruster driver card (Rev. A)
    100051B - Thruster interface card
    100097B - Navigation receiver
    100100D - Navigation detector/counter
    100101B - Navigation preamp
    100104C - Math processor
    100138B - Navigation interface
    100095C - MBM interface

68000 Computer Systems:

    100187D - Main CPU
    100184D - Support CPU
    100148D - UART
    100149D - ROM/RAM
    100154aD - Application Card
    100177C - Video display card
    100181C - Video frame digitizer
    100165C - 8mm camera interface
    100190C - Sonar drawing application card
    100176B - Battery system charger

APPENDIX H

EAVE SOFTWARE LIST

Computer	Software Package
68000 Command	A. VOS
	B. Monitor
	C. Controller Task
	D. Thruster Task
	E. Bubble Task
6100 Navigation	A. SIMNV1
	B. M56XB
	C. NAVC02
	D. NAV46
6100 Thruster	COMC18.PAL
6100 MBM	BUB.PAL

END

FILMED

1-85

DTIC